1. Introduction to Fundamentals of Signal Processing
2. Foundations
    1. Signals Represent Intormation(Thu)
    2. Introduction to Systems
    3. Discrete-Time Signals and Systems
    4. Systems in the Time-Domain
    5. Discrete Time Convolution
    6. Review of Linear Algebra
    7. Hilbert Spaces
    8. Signal Expansions
    9. Introduction to Fourier Analysis
    10. Continuous Time Fourier Transform (CTFT)
    11. Discrete Time Fourier Transform (DTFT)
    12. DFT as a Matrix Operation
    13. The FFT Algorithm
3. Sampling and Frequency Analysis
    1. Introduction
    2. Proof
    3. Illustrations
    4. Sampling and reconstruction with Matlab
    5. Systems view of sampling and reconstruction
    6. Sampling CT Signals: A Frequency Domain Perspective
    7. The DFT: Frequency Domain with a Computer Analysis
    8. Discrete-Time Processing of CT Signals
    9. Short Time Fourier Transform
    10. Spectrograms
    11. Filtering with the DFT
    12. Image Restoration Basics
4. Digital Filtering
    1. Difference Equation
    2. The Z Transform: Definition

3. Table of Common z-Transforms
4. Understanding Pole/Zero Plots on the Z-Plane
5. Filtering in the Frequency Domain
6. Linear-Phase FIR Filters
7. Filter Structures
8. Overview of Digital Filter Design
9. Window Design Method
10. Frequency Sampling Design Method for FIR filters
11. Parks-McClellan FIR Filter Design
12. FIR Filter Design using MATLAB
13. MATLAB FIR Filter Design Exercise

5. Statistical and Adaptive Signal Processing
    1. Introduction to Random Signals and Processes
    2. Stationary and Nonstationary Random Processes
    3. Random Processes: Mean and Variance
    4. Correlation and Covariance of a Random Signal
    5. Autocorrelation of Random Processes
    6. Crosscorrelation of Random Processes
    7. Introduction to Adaptive Filters
    8. Discrete-Time, Causal Wiener Filter
    9. Practical Issues in Wiener Filter Implementation
    10. Quadratic Minimization and Gradient Descent
    11. The LMS Adaptive Filter Algorithm
    12. First Order Convergence Analysis of the LMS Algorithm
    13. Adaptive Equalization

Introduction to Fundamentals of Signal Processing

What is Digital Signal Processing?

To understand what is **Digital Signal Processing (DSP)** let's examine what does each of its words mean. "**Signal**" is any physical quantity that carries information. "**Processing**" is a series of steps or operations to achieve a particular end. It is easy to see that **Signal Processing** is used everywhere to extract information from signals or to convert information-carrying signals from one form to another. For example, our brain and ears take input speech signals, and then process and convert them into meaningful words. Finally, the word "**Digital**" in Digital Signal Processing means that the process is done by computers, microprocessors, or logic circuits.

The field DSP has expanded significantly over that last few decades as a result of rapid developments in computer technology and integrated-circuit fabrication. Consequently, DSP has played an increasingly important role in a wide range of disciplines in science and technology. Research and development in DSP are driving advancements in many high-tech areas including telecommunications, multimedia, medical and scientific imaging, and human-computer interaction.

To illustrate the digital revolution and the impact of DSP, consider the development of digital cameras. Traditional film cameras mainly rely on physical properties of the optical lens, where higher quality requires bigger and larger system, to obtain good images. When digital cameras were first introduced, their quality were inferior compared to film cameras. But as microprocessors become more powerful, more sophisticated DSP algorithms have been developed for digital cameras to correct optical defects and improve the final image quality. Thanks to these developments, the quality of consumer-grade digital cameras has now surpassed the equivalence in film cameras. As further developments for digital cameras attached to cell phones (cameraphones), where due to small size requirements of the lenses, these cameras rely on DSP power to provide good images. Essentially, digital camera technology uses computational power to overcome physical limitations. We can find the similar trend

happens in many other applications of DSP such as digital communications, digital imaging, digital television, and so on.

In summary, DSP has foundations on Mathematics, Physics, and Computer Science, and can provide the key enabling technology in numerous applications.

Overview of Key Concepts in Digital Signal Processing

The two main characters in DSP are **signals** and **systems**. A **signal** is defined as any physical quantity that varies with one or more independent variables such as time (one-dimensional signal), or space (2-D or 3-D signal). Signals exist in several types. In the real-world, most of signals are **continuous-time** or **analog signals** that have values continuously at every value of time. To be processed by a computer, a continuous-time signal has to be first **sampled** in time into a **discrete-time signal** so that its values at a discrete set of time instants can be stored in computer memory locations. Furthermore, in order to be processed by logic circuits, these signal values have to be **quantized** in to a set of discrete values, and the final result is called a **digital signal**. When the quantization effect is ignored, the terms discrete-time signal and digital signal can be used interchangeability.

In signal processing, a **system** is defined as a process whose input and output are signals. An important class of systems is the class of **linear time-invariant** (or **shift-invariant**) **systems**. These systems have a remarkable property is that each of them can be completely characterized by an **impulse response function** (sometimes is also called as **point spread function**), and the system is defined by a **convolution** (also referred to as a **filtering**) operation. Thus, a linear time-invariant system is equivalent to a (linear) **filter**. Linear time-invariant systems are classified into two types, those that have **finite-duration impulse response (FIR)** and those that have an **infinite-duration impulse response (IIR)**.

A signal can be viewed as a **vector** in a **vector space**. Thus, **linear algebra** provides a powerful framework to study signals and linear systems. In particular, given a vector space, each signal can be represented (or expanded) as a **linear combination of elementary signals**. The most

important **signal expansions** are provided by the **Fourier transforms**. The Fourier transforms, as with general transforms, are often used effectively to transform a problem from one domain to another domain where it is much easier to solve or analyze. The two domains of a Fourier transform have physical meaning and are called the **time domain** and the **frequency domain**.

Sampling, or the conversion of **continuous-domain real-life signals** to **discrete numbers** that can be processed by computers, is the essential bridge between the analog and the digital worlds. It is important to understand the connections between signals and systems in the real world and inside a computer. These connections are convenient to analyze in the frequency domain. Moreover, many signals and systems are specified by their **frequency characteristics**.

Because any **linear time-invariant system** can be characterized as a **filter**, the design of such systems boils down to the design the associated filters. Typically, in the **filter design** process, we determine the coefficients of an FIR or IIR filter that closely approximates the desired **frequency response** specifications. Together with Fourier transforms, the **z-transform** provides an effective tool to analyze and design digital filters.

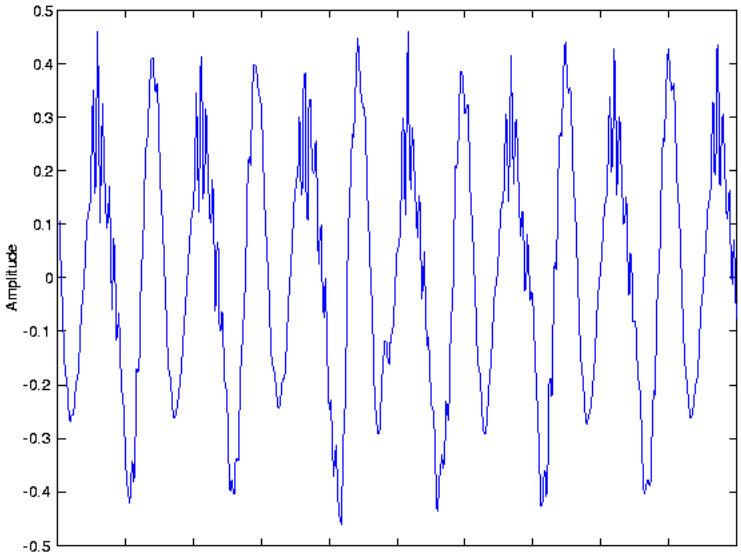
In many applications, signals are conveniently described via **statistical models** as **random signals**. It is remarkable that optimum linear filters (in the sense of **minimum mean-square error**), so called **Wiener filters**, can be determined using only **second-order statistics** (**autocorrelation** and **crosscorrelation** functions) of a **stationary process**. When these statistics cannot be specified beforehand or change over time, we can employ **adaptive filters**, where the filter coefficients are adapted to the signal statistics. The most popular algorithm to adaptively adjust the filter coefficients is the **least-mean square (LMS)** algorithm.

Signals Represent Intormation(Thu)
A brief discussion of information and signals. This module includes an introduction to the notion of continuous and discrete-time signals.

Whether analog or digital, information is represented by the fundamental quantity in electrical engineering: the **signal**. Stated in mathematical terms, **a signal is merely a function**. Analog signals are continuous-valued; digital signals are discrete-valued. The independent variable of the signal could be time (speech, for example), space (images), or the integers (denoting the sequencing of letters and numbers in the football score).

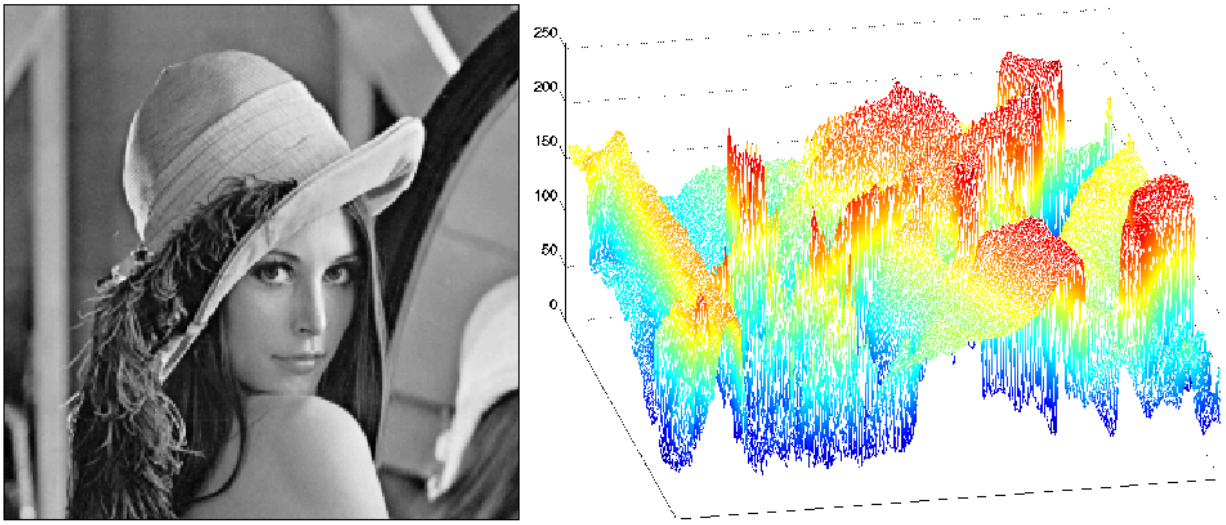
Analog Signals

Analog signals are usually signals defined over continuous independent variable(s). Speech is produced by your vocal cords exciting acoustic resonances in your vocal tract. The result is pressure waves propagating in the air, and the speech signal thus corresponds to a function having independent variables of space and time and a value corresponding to air pressure: $s(\boldsymbol{x},t)$ (Here we use vector notation $\boldsymbol{x}$ to denote spatial coordinates). When you record someone talking, you are evaluating the speech signal at a particular spatial location, $\boldsymbol{x}_0$ say. An example of the resulting waveform $s(\boldsymbol{x}_0,t)$ is shown in this figure.
Speech Example



A speech signal's amplitude relates to tiny air pressure variations. Shown is a recording of the vowel "e" (as in "speech").

Photographs are static, and are continuous-valued signals defined over space. Black-and-white images have only one value at each point in space, which amounts to its optical reflection properties. In [link], an image is shown, demonstrating that it (and all other images as well) are functions of two independent spatial variables.
Lena



On the left is the classic **Lena** image, which is used ubiquitously as a test image. It contains straight and curved lines, complicated texture, and a face. On the right is a perspective display of the Lena image as a signal: a function of two spatial variables. The colors merely help show what signal values are about the same size. In this image, signal values range between 0 and 255; why is that?

Color images have values that express how reflectivity depends on the optical spectrum. Painters long ago found that mixing together combinations of the so-called primary colors--red, yellow and blue--can produce very realistic color images. Thus, images today are usually thought of as having three values at every point in space, but a different set of colors is used: How much of red, **green** and blue is present. Mathematically, color pictures are multivalued--vector-valued--signals: $s(\boldsymbol{x}) = (r(\boldsymbol{x})g(\boldsymbol{x})b(\boldsymbol{x}))^T$.

Interesting cases abound where the analog signal depends not on a continuous variable, such as time, but on a discrete variable. For example, temperature readings taken every hour have continuous--analog--values, but the signal's independent variable is (essentially) the integers.

Digital Signals

The word "digital" means discrete-valued and implies the signal has an integer-valued independent variable. Digital information includes numbers and symbols (characters typed on the keyboard, for example). Computers rely on the digital representation of information to manipulate and transform information. Symbols do not have a numeric value, and each is represented by a unique number. The ASCII character code has the upper- and lowercase characters, the numbers, punctuation marks, and various other symbols represented by a seven-bit integer. For example, the ASCII code represents the letter **a** as the number 97 and the letter **A** as 65. [link] shows the international convention on associating characters with integers.

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ac
08	bs	09	ht	0A	nl	0B	vt	0C	np	0D	cr	0E	so

10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	sy
18	car	19	em	1A	sub	1B	esc	1C	fs	1D	gs	1E	rs
20	sp	21	!	22	"	23	#	24	$	25	%	26	&
28	(	29	)	2A	*	2B	+	2C	,	2D	-	2E	.
30	0	31	1	32	2	33	3	34	4	35	5	36	6
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>
40	@	41	A	42	B	43	C	44	D	45	E	46	F
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N
50	P	51	Q	52	R	53	S	54	T	55	U	56	V
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^
60	'	61	a	62	b	63	c	64	d	65	e	66	f
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n
70	p	71	q	72	r	73	s	74	t	75	u	76	v
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	~

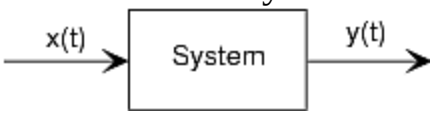
ASCII Table The ASCII translation table shows how standard keyboard characters are represented by integers. In pa
table displays first the so-called 7-bit code (how many characters in a seven-bit code?), then the character the numbe
numeric codes are represented in hexadecimal (base-16) notation. Mnemonic characters correspond to control chara
may be familiar (like **cr** for carriage return) and some not (**bel** means a "bell").

Introduction to Systems

Introduction to the concept of a system, which is a mechanism for manipulating signals. Feedback concepts and superpositions are also briefly mentioned.

Signals are manipulated by systems. Mathematically, we represent what a system does by the notation $y(t) = S(x(t))$, with x representing the input signal and y the output signal.

Definition of a system



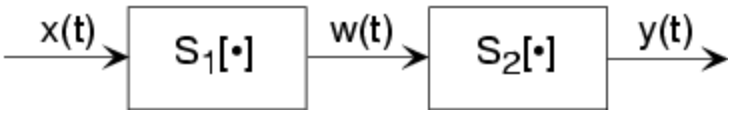
The system depicted has input $x(t)$ and output $y(t)$. Mathematically, systems operate on function(s) to produce other function(s). In many ways, systems are like functions, rules that yield a value for the dependent variable (our output signal) for each value of its independent variable (its input signal). The notation $y(t) = S(x(t))$ corresponds to this block diagram. We term $S(\cdot)$ the input-output relation for the system.

This notation mimics the mathematical symbology of a function: A system's input is analogous to an independent variable and its output the dependent variable. For the mathematically inclined, a system is a **functional**: a function of a function (signals are functions).

Simple systems can be connected together--one system's output becomes another's input--to accomplish some overall design. Interconnection topologies can be quite complicated, but usually consist of weaves of three basic interconnection forms.

Cascade Interconnection

cascade

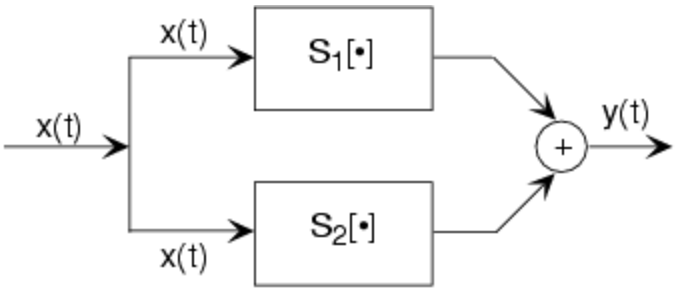


The most rudimentary ways of interconnecting systems are shown in the figures in this section. This is the cascade configuration.

The simplest form is when one system's output is connected only to another's input. Mathematically, $w(t) = S_1(x(t))$, and $y(t) = S_2(w(t))$, with the information contained in $x(t)$ processed by the first, then the second system. In some cases, the ordering of the systems matter, in others it does not. For example, in the fundamental model of communication the ordering most certainly matters.

Parallel Interconnection

parallel

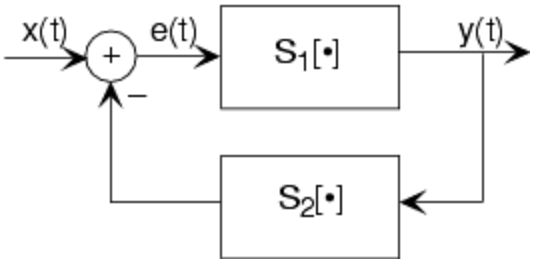


The parallel configuration.

A signal $x(t)$ is routed to two (or more) systems, with this signal appearing as the input to all systems simultaneously and with equal strength. Block diagrams have the convention that signals going to more than one system are not split into pieces along the way. Two or more systems operate on $x(t)$ and their outputs are added together to create the output $y(t)$. Thus, $y(t) = S_1(x(t)) + S_2(x(t))$, and the information in $x(t)$ is processed separately by both systems.

Feedback Interconnection

feedback



The feedback configuration.

The subtlest interconnection configuration has a system's output also contributing to its input. Engineers would say the output is "fed back" to the

input through system 2, hence the terminology. The mathematical statement of the feedback interconnection is that the feed-forward system produces the output: $y(t) = S_1(e(t))$. The input $e(t)$ equals the input signal minus the output of some other system's output to $y(t)$: $e(t) = x(t) - S_2(y(t))$. Feedback systems are omnipresent in control problems, with the error signal used to adjust the output to achieve some condition defined by the input (controlling) signal. For example, in a car's cruise control system, $x(t)$ is a constant representing what speed you want, and $y(t)$ is the car's speed as measured by a speedometer. In this application, system 2 is the identity system (output equals input).

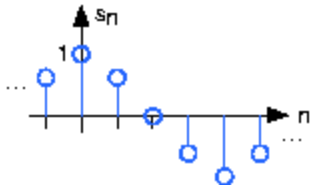
Discrete-Time Signals and Systems
(Blank Abstract)

Mathematically, analog signals are functions having as their independent variables continuous quantities, such as space and time. Discrete-time signals are functions defined on the integers; they are sequences. As with analog signals, we seek ways of decomposing discrete-time signals into simpler components. Because this approach leads to a better understanding of signal structure, we can exploit that structure to represent information (create ways of representing information with signals) and to extract information (retrieve the information thus represented). For symbolic-valued signals, the approach is different: We develop a common representation of all symbolic-valued signals so that we can embody the information they contain in a unified way. From an information representation perspective, the most important issue becomes, for both real-valued and symbolic-valued signals, efficiency: what is the most parsimonious and compact way to represent information so that it can be extracted later.

Real- and Complex-valued Signals

A discrete-time signal is represented symbolically as $s(n)$, where $n = \{\ldots, -1, 0, 1, \ldots\}$.

Cosine



The discrete-time cosine signal is plotted as a stem plot. Can you find the formula for this signal?

We usually draw discrete-time signals as stem plots to emphasize the fact they are functions defined only on the integers. We can delay a discrete-time signal by an integer just as with analog ones. A signal delayed by m samples has the expression $s(n-m)$.

Complex Exponentials

The most important signal is, of course, the **complex exponential sequence**.
Equation:

$$s(n) = e^{i2\pi fn}$$

Note that the frequency variable f is dimensionless and that adding an integer to the frequency of the discrete-time complex exponential has no effect on the signal's value.
Equation:

$$\begin{aligned} e^{i2\pi(f+m)n} &= e^{i2\pi fn}e^{i2\pi mn} \\ &= e^{i2\pi fn} \end{aligned}$$

This derivation follows because the complex exponential evaluated at an integer multiple of 2π equals one. Thus, we need only consider frequency to have a value in some unit-length interval.

Sinusoids

Discrete-time sinusoids have the obvious form $s(n) = A\cos(2\pi fn + \varphi)$. As opposed to analog complex exponentials and sinusoids that can have their frequencies be any real value, frequencies of their discrete-time counterparts yield unique waveforms **only** when f lies in the interval

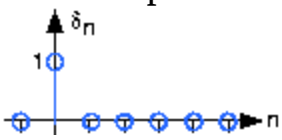
$\left(-\frac{1}{2}, \frac{1}{2}\right]$. This choice of frequency interval is arbitrary; we can also choose the frequency to lie in the interval $[0, 1)$. How to choose a unit-length interval for a sinusoid's frequency will become evident later.

Unit Sample

The second-most important discrete-time signal is the **unit sample**, which is defined to be
Equation:

$$\delta(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases}$$

Unit sample



The unit sample.

Examination of a discrete-time signal's plot, like that of the cosine signal shown in [link], reveals that all signals consist of a sequence of delayed and scaled unit samples. Because the value of a sequence at each integer m is denoted by $s(m)$ and the unit sample delayed to occur at m is written $\delta(n - m)$, we can decompose **any** signal as a sum of unit samples delayed to the appropriate location and scaled by the signal value.
Equation:

$$s(n) = \sum_{m=-\infty}^{\infty} s(m)\delta(n - m)$$

This kind of decomposition is unique to discrete-time signals, and will prove useful subsequently.

Unit Step

The **unit step** in discrete-time is well-defined at the origin, as opposed to the situation with analog signals.
Equation:

$$u(n) = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{if } n < 0 \end{cases}$$

Symbolic Signals

An interesting aspect of discrete-time signals is that their values do not need to be real numbers. We do have real-valued discrete-time signals like the sinusoid, but we also have signals that denote the sequence of characters typed on the keyboard. Such characters certainly aren't real numbers, and as a collection of possible signal values, they have little mathematical structure other than that they are members of a set. More formally, each element of the **symbolic-valued** signal $s(n)$ takes on one of the values $\{a_1, \ldots, a_K\}$ which comprise the **alphabet** A. This technical terminology does not mean we restrict symbols to being members of the English or Greek alphabet. They could represent keyboard characters, bytes (8-bit quantities), integers that convey daily temperature. Whether controlled by software or not, discrete-time systems are ultimately constructed from digital circuits, which consist **entirely** of analog circuit elements. Furthermore, the transmission and reception of discrete-time signals, like e-mail, is accomplished with analog signals and systems. Understanding how discrete-time and analog signals and systems intertwine is perhaps the main goal of this course.

Discrete-Time Systems

Discrete-time systems can act on discrete-time signals in ways similar to those found in analog signals and systems. Because of the role of software

in discrete-time systems, many more different systems can be envisioned and "constructed" with programs than can be with analog signals. In fact, a special class of analog signals can be converted into discrete-time signals, processed with software, and converted back into an analog signal, all without the incursion of error. For such signals, systems can be easily produced in software, with equivalent analog realizations difficult, if not impossible, to design.

Systems in the Time-Domain

A discrete-time signal $s(n)$ is **delayed** by n_0 samples when we write $s(n - n_0)$, with $n_0 > 0$. Choosing n_0 to be negative advances the signal along the integers. As opposed to analog delays, discrete-time delays can **only** be integer valued. In the frequency domain, delaying a signal corresponds to a linear phase shift of the signal's discrete-time Fourier transform: $s(n - n_0) \leftrightarrow e^{-(i2\pi f n_0)} S\left(e^{i2\pi f}\right)$.

Linear discrete-time systems have the superposition property.
Equation:

Superposition

$$S(a_1 x_1(n) + a_2 x_2(n)) = a_1 S(x_1(n)) + a_2 S(x_2(n))$$

A discrete-time system is called **shift-invariant** (analogous to time-invariant analog systems) if delaying the input delays the corresponding output.
Equation:

Shift-Invariant

$$\text{If } \; S(x(n)) = y(n), \text{ Then } \; S(x(n - n_0)) = y(n - n_0)$$

We use the term shift-invariant to emphasize that delays can only have integer values in discrete-time, while in analog signals, delays can be arbitrarily valued.

We want to concentrate on systems that are both linear and shift-invariant. It will be these that allow us the full power of frequency-domain analysis and implementations. Because we have no physical constraints in "constructing" such systems, we need only a mathematical specification. In analog systems, the differential equation specifies the input-output relationship in the time-domain. The corresponding discrete-time specification is the **difference equation**.
Equation:

The Difference Equation

$$y(n) = a_1 y(n-1) + \ldots + a_p y(n-p) + b_0 x(n) + b_1 x(n-1) + \ldots + b_q x(n-q)$$

Here, the output signal $y(n)$ is related to its **past** values $y(n - l)$, $l = \{1, \ldots, p\}$, and to the current and past values of the input signal $x(n)$. The system's characteristics are determined by the choices for the number of coefficients p and q and the coefficients' values $\{a_1, \ldots, a_p\}$ and $\{b_0, b_1, \ldots, b_q\}$.

Note: There is an asymmetry in the coefficients: where is a_0 ? This coefficient would multiply the $y(n)$ term in the difference equation. We have essentially divided the equation by it, which does not change the input-output relationship. We have thus created the convention that a_0 is always one.

As opposed to differential equations, which only provide an **implicit** description of a system (we must somehow solve the differential equation), difference equations provide an **explicit** way of computing the output for any input. We simply express the difference equation by a program that calculates each output from the previous output values, and the current and previous inputs.

Discrete Time Convolution
Convolution is a concept that extends to all systems that are both linear and time-invariant (LTI). It will become apparent in this discussion that this condition is necessary by demonstrating how linearity and time-invariance give rise to convolution.

Introduction

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output a system produces for a given input signal. It can be shown that a linear time invariant system is completely characterized by its impulse response. The sifting property of the discrete time impulse function tells us that the input signal to a system can be represented as a sum of scaled and shifted unit impulses. Thus, by linearity, it would seem reasonable to compute of the output signal as the sum of scaled and shifted unit impulse responses. That is exactly what the operation of convolution accomplishes. Hence, convolution can be used to determine a linear time invariant system's output from knowledge of the input and the impulse response.

Convolution and Circular Convolution

Convolution

Operation Definition

Discrete time convolution is an operation on two discrete time signals defined by the integral
Equation:

$$(f*g)[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k]$$

for all signals f, g defined on $\mathbb{Z}$. It is important to note that the operation of convolution is commutative, meaning that
Equation:

$$f*g = g*f$$

for all signals f, g defined on $\mathbb{Z}$. Thus, the convolution operation could have been just as easily stated using the equivalent definition
Equation:

$$(f*g)[n] = \sum_{k=-\infty}^{\infty} f[n-k]g[k]$$

for all signals f, g defined on $\mathbb{Z}$. Convolution has several other important properties not listed here but explained and derived in a later module.

Definition Motivation

The above operation definition has been chosen to be particularly useful in the study of linear time invariant systems. In order to see this, consider a linear time invariant system H with unit impulse response h. Given a system input signal x we would like to compute the system output signal $H(x)$. First, we note that the input can be expressed as the convolution
Equation:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

by the sifting property of the unit impulse function. By linearity
Equation:

$$H(x[n]) = \sum_{k=-\infty}^{\infty} x[k]H(\delta[n-k]).$$

Since $H(\delta[n-k])$ is the shifted unit impulse response $h[n-k]$, this gives the result
Equation:

$$H\left(x\left[n\right]\right) = \sum_{k=-\infty}^{\infty} x\left[k\right] h\left[n-k\right] = \left(x^{*} h\right)\left[n\right].$$

Hence, convolution has been defined such that the output of a linear time invariant system is given by the convolution of the system input with the system unit impulse response.

Graphical Intuition

It is often helpful to be able to visualize the computation of a convolution in terms of graphical processes. Consider the convolution of two functions f, g given by
Equation:

$$\left(f^{*} g\right)\left[n\right] = \sum_{k=-\infty}^{\infty} f\left[k\right] g\left[n-k\right] = \sum_{k=-\infty}^{\infty} f\left[n-k\right] g\left[k\right].$$

The first step in graphically understanding the operation of convolution is to plot each of the functions. Next, one of the functions must be selected, and its plot reflected across the $k = 0$ axis. For each real n, that same function must be shifted left by n. The point-wise product of the two resulting plots is then computed, and then all of the values are summed.

Example:
Recall that the impulse response for a discrete time echoing feedback system with gain a is

Equation:

$$h\left[n\right] = a^{n}u\left[n\right],$$

and consider the response to an input signal that is another exponential
Equation:

$$x\left[n\right] = b^{n}u\left[n\right].$$

We know that the output for this input is given by the convolution of the impulse response with the input signal
Equation:

$$y[n] = x[n]*h[n].$$

We would like to compute this operation by beginning in a way that minimizes the algebraic complexity of the expression. However, in this case, each possible choice is equally simple. Thus, we would like to compute
Equation:

$$y\left[n\right] = \sum_{k=-\infty}^{\infty} a^{k}u\left[k\right]b^{n-k}u\left[n-k\right].$$

The step functions can be used to further simplify this sum. Therefore,
Equation:

$$y[n] = 0$$

for $n < 0$ and
Equation:

$$y\left[n\right] = \sum_{k=0}^{n} \left[ab\right]^{k}$$

for $n \geq 0$. Hence, provided $ab \neq 1$, we have that
Equation:

$$y[n] = \begin{cases} 0 & n < 0 \\ \frac{1-(ab)^{n+1}}{1-(ab)} & n \geq 0 \end{cases}.$$

Circular Convolution

Discrete time circular convolution is an operation on two finite length or periodic discrete time signals defined by the sum
Equation:

$$(f \circledast g)[n] = \sum_{k=0}^{N-1} \widehat{f}[k] \hat{g}[n-k]$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$ where $\widehat{f}, \hat{g}$ are periodic extensions of f and g. It is important to note that the operation of circular convolution is commutative, meaning that
Equation:

$$f \circledast g = g \circledast f$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$. Thus, the circular convolution operation could have been just as easily stated using the equivalent definition
Equation:

$$(f \circledast g)[n] = \sum_{k=0}^{N-1} \widehat{f}[n-k] \hat{g}[k]$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$ where $\widehat{f}, \hat{g}$ are periodic extensions of f and g. Circular convolution has several other important properties not listed here but explained and derived in a later module.

Alternatively, discrete time circular convolution can be expressed as the sum of two summations given by
Equation:

$$(f \circledast g)[n] = \sum_{k=0}^{n} f[k] g[n-k] + \sum_{k=n+1}^{N-1} f[k] g[n-k+N]$$

for all signals f, g defined on $\mathbb{Z}[0, N-1]$.

Meaningful examples of computing discrete time circular convolutions in the time domain would involve complicated algebraic manipulations dealing with the wrap around behavior, which would ultimately be more confusing than helpful. Thus, none will be provided in this section. Of course, example computations in the time domain are easy to program and demonstrate. However, disrete time circular convolutions are more easily computed using frequency domain tools as will be shown in the discrete time Fourier series section.

Definition Motivation

The above operation definition has been chosen to be particularly useful in the study of linear time invariant systems. In order to see this, consider a linear time invariant system H with unit impulse response h. Given a periodic system input signal x we would like to compute the system output signal $H(x)$. First, we note that the input can be expressed as the circular convolution
Equation:

$$x[n] = \sum_{k=0}^{N-1} \widehat{x}[k] \hat{\delta}[n-k]$$

by the sifting property of the unit impulse function. By linearity,
Equation:

$$H\left(x\left[n\right]\right) = \sum_{k=0}^{N-1} \widehat{x}\left[k\right] H\left(\hat{\delta}\left[n-k\right]\right).$$

Since $H(\delta[n-k])$ is the shifted unit impulse response $h[n-k]$, this gives the result

Equation:

$$H\left(x\left[n\right]\right) = \sum_{k=0}^{N-1} \widehat{x}\left[k\right] \widehat{h}\left[n-k\right] = \left(x \circledast h\right)\left[n\right].$$

Hence, circular convolution has been defined such that the output of a linear time invariant system is given by the convolution of the system input with the system unit impulse response.

Graphical Intuition

It is often helpful to be able to visualize the computation of a circular convolution in terms of graphical processes. Consider the circular convolution of two finite length functions f, g given by

Equation:

$$\left(f \circledast g\right)\left[n\right] = \sum_{k=0}^{N-1} \widehat{f}\left[k\right] \hat{g}\left[n-k\right] = \sum_{k=0}^{N-1} \widehat{f}\left[n-k\right] \hat{g}\left[k\right].$$

The first step in graphically understanding the operation of convolution is to plot each of the periodic extensions of the functions. Next, one of the functions must be selected, and its plot reflected across the $k = 0$ axis. For each $n \in \mathbb{Z}[0, N-1]$, that same function must be shifted left by n. The point-wise product of the two resulting plots is then computed, and finally all of these values are summed.

Interactive Element

Interact (when online) with the Mathematica CDF demonstrating Discrete Linear Convolution. To download, right click and save file as .cdf

Convolution Summary

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output signal of a linear time invariant system for a given input signal with knowledge of the system's unit impulse response. The operation of discrete time convolution is defined such that it performs this function for infinite length discrete time signals and systems. The operation of discrete time circular convolution is defined such that it performs this function for finite length and periodic discrete time signals. In each case, the output of the system is the convolution or circular convolution of the input signal with the unit impulse response.

Review of Linear Algebra

Vector spaces are the principal object of study in linear algebra. A vector space is always defined with respect to a field of scalars.

Fields

A field is a set F equipped with two operations, addition and mulitplication, and containing two special members 0 and 1 ($0 \neq 1$), such that for all $\{a, b, c\} \in F$

1. 1. $(a + b) \in F$
   2. $a + b = b + a$
   3. $(a + b) + c = a + (b + c)$
   4. $a + 0 = a$
   5. there exists $-a$ such that $a + -a = 0$

2. 1. $ab \in F$
   2. $ab = ba$
   3. $(ab)c = a\,(bc)$
   4. $a \cdot 1 = a$
   5. there exists a^{-1} such that $aa^{-1} = 1$

3. $a\,(b + c) = ab + ac$

More concisely

1. F is an abelian group under addition
2. F is an abelian group under multiplication
3. multiplication distributes over addition

Examples

$\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$

Vector Spaces

Let F be a field, and V a set. We say V **is a vector space over** F if there exist two operations, defined for all $a \in F$, $\boldsymbol{u} \in V$ and $\boldsymbol{v} \in V$:

- vector addition: $(\boldsymbol{u}, \boldsymbol{v}) \to (\boldsymbol{u} + \boldsymbol{v}) \in V$
- scalar multiplication: $(a, \boldsymbol{v}) \to a\boldsymbol{v} \in V$

and if there exists an element denoted $\mathbf{0} \in V$, such that the following hold for all $a \in F$, $b \in F$, and $\boldsymbol{u} \in V$, $\boldsymbol{v} \in V$, and $\boldsymbol{w} \in V$

1. 1. $\boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}) = (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}$
   2. $\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}$
   3. $\boldsymbol{u} + \mathbf{0} = \boldsymbol{u}$
   4. there exists $-\boldsymbol{u}$ such that $\boldsymbol{u} + -\boldsymbol{u} = \mathbf{0}$

2. 1. $a\,(\boldsymbol{u} + \boldsymbol{v}) = a\boldsymbol{u} + a\boldsymbol{v}$
   2. $(a + b)\boldsymbol{u} = a\boldsymbol{u} + b\boldsymbol{u}$
   3. $(ab)\boldsymbol{u} = a\,(b\boldsymbol{u})$
   4. $1 \cdot \boldsymbol{u} = \boldsymbol{u}$

More concisely,

1. V is an abelian group under plus
2. Natural properties of scalar multiplication

Examples

- $\mathbb{R}^N$ is a vector space over $\mathbb{R}$
- $\mathbb{C}^N$ is a vector space over $\mathbb{C}$
- $\mathbb{C}^N$ is a vector space over $\mathbb{R}$
- $\mathbb{R}^N$ is **not** a vector space over $\mathbb{C}$

The elements of V are called **vectors**.

Euclidean Space

Throughout this course we will think of a signal as a vector

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & \dots & x_N \end{pmatrix}^T$$

The samples $\{x_i\}$ could be samples from a finite duration, continuous time signal, for example.

A signal will belong to one of two vector spaces:

Real Euclidean space

$\boldsymbol{x} \in \mathbb{R}^N$ (over $\mathbb{R}$)

Complex Euclidean space

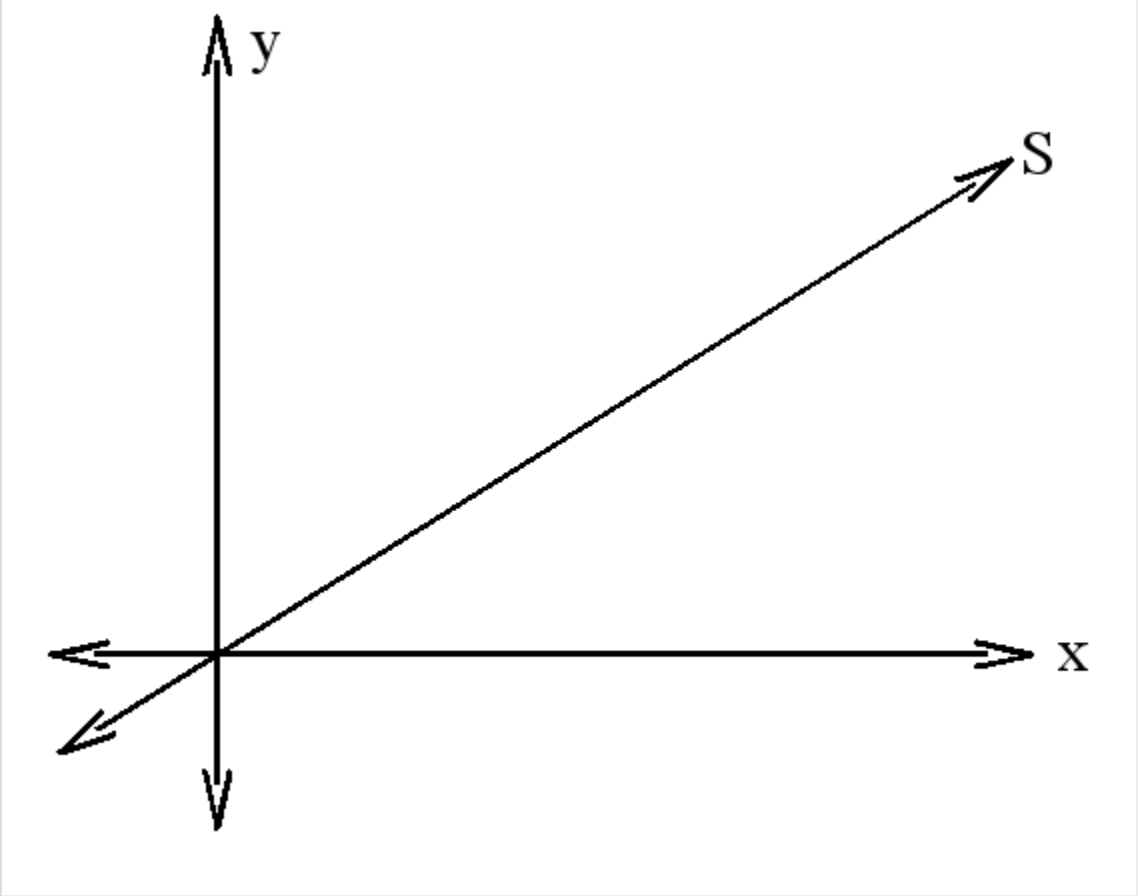
$\boldsymbol{x} \in \mathbb{C}^N$ (over $\mathbb{C}$)

Subspaces

Let V be a vector space over F.

A subset $S \subseteq V$ is called a **subspace** of V if S is a vector space over F in its own right.

Example:
$V = \mathbb{R}^2$, $F = \mathbb{R}$, $S =$ any line though the origin.



S is any line through the origin.

Are there other subspaces?

$S \subseteq V$ is a subspace if and only if for all $a \in F$ and $b \in F$ and for all $\boldsymbol{s} \in S$ and $\boldsymbol{t} \in S$, $(a\boldsymbol{s} + b\boldsymbol{t}) \in S$

Linear Independence

Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k \in V$.

We say that these vectors are **linearly dependent** if there exist scalars $a_1, \ldots, a_k \in F$ such that

Equation:

$$\sum_{i=1}^{k} a_i u_i = \mathbf{0}$$

and at least one $a_i \neq 0$.

If [link] only holds for the case $a_1 = \ldots = a_k = 0$, we say that the vectors are **linearly independent**.

Example:

$$1\begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} - 2\begin{pmatrix} -2 \\ 3 \\ 0 \end{pmatrix} + 1\begin{pmatrix} -5 \\ 7 \\ -2 \end{pmatrix} = \mathbf{0}$$

so these vectors are linearly dependent in $\mathbb{R}^3$.

Spanning Sets

Consider the subset $S = \{v_1, v_2, \ldots, v_k\}$. Define the **span** of S

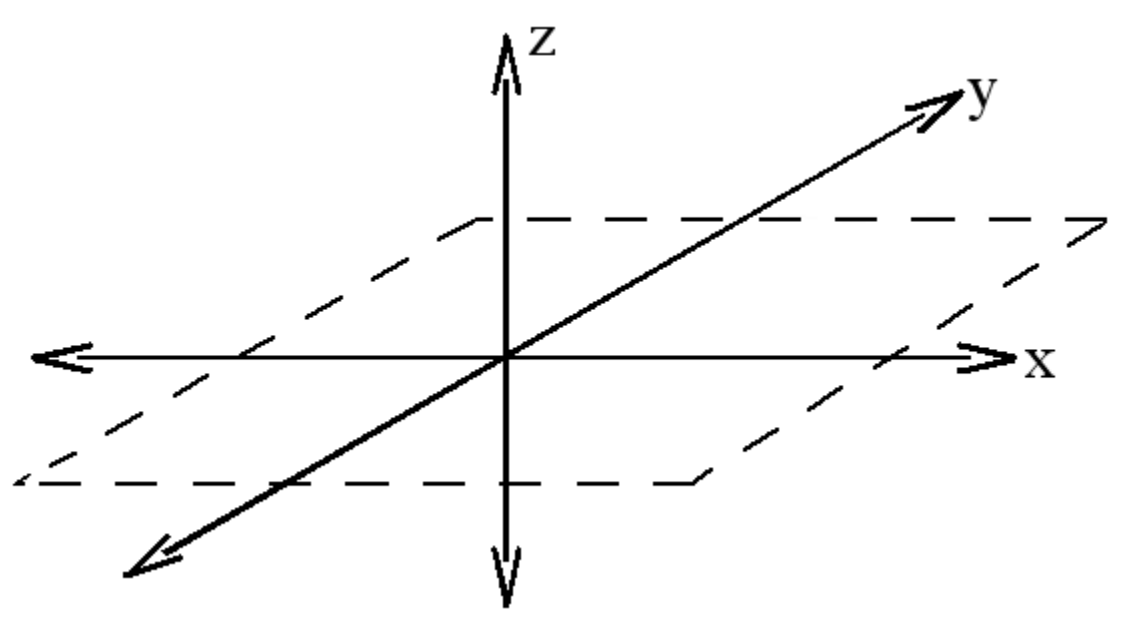
$$< S > \equiv \text{span}\,(S) \equiv \left\{ \sum_{i=1}^{k} a_i v_i \,\middle|\, a_i \in F \right\}$$

Fact: $< S >$ is a subspace of V.

Example:

$V = \mathbb{R}^3$, $F = \mathbb{R}$, $S = \{v_1, v_2\}$, $v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \Rightarrow$

$< S >=$ xy-plane.



$< S >$ is the xy-plane.

Aside

If S is infinite, the notions of linear independence and span are easily generalized:

We say S is linearly independent if, for every finite collection $u_1, \ldots, u_k \in S$, (k arbitrary) we have

$$\left(\sum_{i=1}^{k} a_i u_i = \mathbf{0} \right) \Rightarrow \forall i : (a_i = 0)$$

The span of S is

$$< S >= \left\{ \sum_{i=1}^{k} a_i u_i \,\middle|\, a_i \in F \ \wedge \ u_i \in S \ \wedge \ (k < \infty) \right\}$$

Note: In both definitions, we only consider **finite** sums.

Bases

A set $B \subseteq V$ is called a **basis** for V over F if and only if

1. B is linearly independent
2. $< B >= V$

Bases are of fundamental importance in signal processing. They allow us to decompose a signal into building blocks (basis vectors) that are often more easily understood.

Example:
V = (real or complex) Euclidean space, $\mathbb{R}^N$ or $\mathbb{C}^N$.

$$B = \{e_1, \ldots, e_N\} \equiv \text{standard basis}$$

$$e_i = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

where the 1 is in the i^{th} position.

Example:
$V = \mathbb{C}^N$ over $\mathbb{C}$.

$$B = \{u_1, \ldots, u_N\}$$

which is the DFT basis.

$$u_k = \begin{pmatrix} 1 \\ e^{-\left(i2\pi\frac{k}{N}\right)} \\ \vdots \\ e^{-\left(i2\pi\frac{k}{N}(N-1)\right)} \end{pmatrix}$$

where $i = \sqrt{-1}$.

Key Fact

If B is a basis for V, then every $\boldsymbol{v} \in V$ can be written uniquely (up to order of terms) in the form

$$\boldsymbol{v} = \sum_{i=1}^{N} a_i v_i$$

where $a_i \in F$ and $v_i \in B$.

Other Facts

- If S is a linearly independent set, then S can be extended to a basis.
- If $< S >= V$, then S contains a basis.

Dimension

Let V be a vector space with basis B. The dimension of V, denoted $\dim(V)$, is the cardinality of B.

Every vector space has a basis.

Every basis for a vector space has the same cardinality.

$\Rightarrow \dim(V)$ is **well-defined**.

If $\dim(V) < \infty$, we say V is **finite dimensional**.

Examples

vector space	field of scalars	dimension
$\mathbb{R}^N$	$\mathbb{R}$	
$\mathbb{C}^N$	$\mathbb{C}$	
$\mathbb{C}^N$	$\mathbb{R}$	

Every subspace is a vector space, and therefore has its own dimension.

Example:
Suppose $(S = \{u_1, \ldots, u_k\}) \subseteq V$ is a linearly independent set. Then

$$\dim(< S >) =$$

Facts

- If S is a subspace of V, then $\dim(S) \leq \dim(V)$.
- If $\dim(S) = \dim(V) < \infty$, then $S = V$.

Direct Sums

Let V be a vector space, and let $S \subseteq V$ and $T \subseteq V$ be subspaces.

We say V is the **direct sum** of S and T, written $V = S \oplus T$, if and only if for every $\boldsymbol{v} \in V$, there exist unique $\boldsymbol{s} \in S$ and $\boldsymbol{t} \in T$ such that $\boldsymbol{v} = \boldsymbol{s} + \boldsymbol{t}$.

If $V = S \oplus T$, then T is called a **complement** of S.

Example:

$$V = C' = \{f : \mathbb{R} \to \mathbb{R} | f \text{is continuous}\}$$

$$S = \text{even funcitons in} C'$$

$$T = \text{odd funcitons in} C'$$

$$f(t) = \frac{1}{2}\left(f(t) + f(-t)\right) + \frac{1}{2}\left(f(t) - f(-t)\right)$$

If $f = g + h = g' + h'$, $g \in S$ and $g' \in S$, $h \in T$ and $h' \in T$, then $g - g' = h' - h$ is odd and even, which implies $g = g'$ and $h = h'$.

Facts

1. Every subspace has a complement
2. $V = S \oplus T$ if and only if
    1. $S \cap T = \{\mathbf{0}\}$
    2. $< S, T >= V$

3. If $V = S \oplus T$, and $\dim(V) < \infty$, then
   $\dim(V) = \dim(S) + \dim(T)$

Proofs

Invoke a basis.

Norms

Let V be a vector space over F. A norm is a mapping $V \to F$, denoted by $\| \cdot \|$, such that forall $\boldsymbol{u} \in V$, $\boldsymbol{v} \in V$, and $\lambda \in F$

1. $\| \boldsymbol{u} \| > 0$ if $\boldsymbol{u} \neq \boldsymbol{0}$
2. $\| \lambda \boldsymbol{u} \| = |\lambda| \, \| \boldsymbol{u} \|$
3. $\| \boldsymbol{u} + \boldsymbol{v} \| \leq \| \boldsymbol{u} \| + \| \boldsymbol{v} \|$

Examples

Euclidean norms:

$\boldsymbol{x} \in \mathbb{R}^N$:

$$\| \boldsymbol{x} \| = \left(\sum_{i=1}^{N} {x_i}^2 \right)^{\frac{1}{2}}$$

$\boldsymbol{x} \in \mathbb{C}^N$:

$$\| \boldsymbol{x} \| = \left(\sum_{i=1}^{N} \left(|x_i| \right)^2 \right)^{\frac{1}{2}}$$

Induced Metric

Every norm induces a metric on V

$$d(\boldsymbol{u}, \boldsymbol{v}) \equiv \| \boldsymbol{u} - \boldsymbol{v} \|$$

which leads to a notion of "distance" between vectors.

Inner products

Let V be a vector space over F, $F = \mathbb{R}$ or $\mathbb{C}$. An inner product is a mapping $V \times V \to F$, denoted $\langle \cdot, \cdot \rangle$, such that

1. $\langle \boldsymbol{v}, \boldsymbol{v} \rangle \geq 0$, and $\langle \boldsymbol{v}, \boldsymbol{v} \rangle = 0 \Leftrightarrow \boldsymbol{v} = 0$
2. $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \overline{\langle \boldsymbol{v}, \boldsymbol{u} \rangle}$
3. $\langle a\boldsymbol{u} + b\boldsymbol{v}, \boldsymbol{w} \rangle = a \langle (\boldsymbol{u}, \boldsymbol{w}) \rangle + b \langle (\boldsymbol{v}, \boldsymbol{w}) \rangle$

Examples

$\mathbb{R}^N$ over $\mathbb{R}$:

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \left(\boldsymbol{x}^T \boldsymbol{y} \right) = \sum_{i=1}^{N} x_i y_i$$

$\mathbb{C}^N$ over $\mathbb{C}$:

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \left(\boldsymbol{x}^{\mathrm{H}} \boldsymbol{y} \right) = \sum_{i=1}^{N} \overline{x_i} y_i$$

If $\left(\boldsymbol{x} = (x_1 \ldots x_N)^T \right) \in \mathbb{C}$, then

$$\boldsymbol{x}^{\mathrm{H}} \equiv \begin{pmatrix} \overline{x_1} \\ \vdots \\ \overline{x_N} \end{pmatrix}^T$$

is called the "Hermitian," or "conjugate transpose" of $\boldsymbol{x}$.

Triangle Inequality

If we define $\| \boldsymbol{u} \| = \langle \boldsymbol{u}, \boldsymbol{u} \rangle$, then

$$\| \boldsymbol{u} + \boldsymbol{v} \| \leq \| \boldsymbol{u} \| + \| \boldsymbol{v} \|$$

Hence, every inner product induces a norm.

Cauchy-Schwarz Inequality

For all $\boldsymbol{u} \in V$, $\boldsymbol{v} \in V$,

$$|\langle \boldsymbol{u}, \boldsymbol{v} \rangle| \leq \| \boldsymbol{u} \| \ \ \| \boldsymbol{v} \|$$

In inner product spaces, we have a notion of the angle between two vectors:

$$\left(\angle(\boldsymbol{u}, \boldsymbol{v}) = \arccos\left(\frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\| \boldsymbol{u} \| \ \ \| \boldsymbol{v} \|} \right) \right) \in [0, 2\pi)$$

Orthogonality

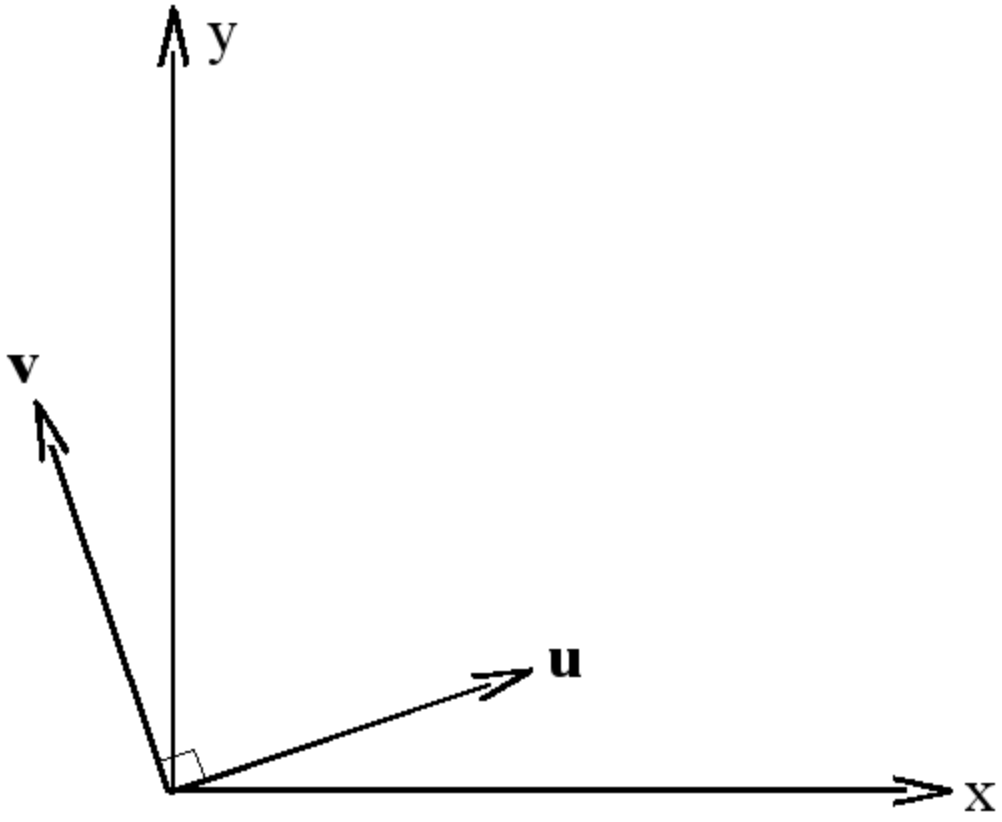
$\boldsymbol{u}$ and $\boldsymbol{v}$ are **orthogonal** if

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = 0$$

Notation: $\boldsymbol{u} \perp \boldsymbol{v}$.

If in addition $\| \boldsymbol{u} \| = \| \boldsymbol{v} \| = 1$, we say $\boldsymbol{u}$ and $\boldsymbol{v}$ are **orthonormal**.

In an orthogonal (orthonormal) **set**, each pair of vectors is orthogonal (orthonormal).



Orthogonal vectors in $\mathbb{R}^2$.

Orthonormal Bases

An Orthonormal basis is a basis $\{v_i\}$ such that

$$\langle v_i, v_i \rangle = \delta_{ij} = \begin{cases} 1 & \text{if } \ i = j \\ 0 & \text{if } \ i \neq j \end{cases}$$

Example:
The standard basis for $\mathbb{R}^N$ or $\mathbb{C}^N$

Example:
The normalized DFT basis

$$u_k = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 \\ e^{-\left(i2\pi \frac{k}{N}\right)} \\ \vdots \\ e^{-\left(i2\pi \frac{k}{N}(N-1)\right)} \end{pmatrix}$$

Expansion Coefficients

If the representation of $\boldsymbol{v}$ with respect to $\{v_i\}$ is

$$\boldsymbol{v} = \sum_i a_i v_i$$

then

$$a_i = \langle v_i, \boldsymbol{v} \rangle$$

Gram-Schmidt

Every inner product space has an orthonormal basis. Any (countable) basis can be made orthogonal by the Gram-Schmidt orthogonalization process.

Orthogonal Compliments

Let $S \subseteq V$ be a subspace. The **orthogonal compliment** S is

$$S^{\perp} = \{\boldsymbol{u} | \, \boldsymbol{u} \in V \ \wedge \ (\langle \boldsymbol{u}, \boldsymbol{v} \rangle = 0) \ \wedge \ \forall \boldsymbol{v} : (\boldsymbol{v} \in S)\}$$

$S^{\perp}$ is easily seen to be a subspace.

If $\dim(v) < \infty$, then $V = S \oplus S^{\perp}$.

Note: If $\dim(v) = \infty$, then in order to have $V = S \oplus S^{\perp}$ we require V to be a **Hilbert Space**.

Linear Transformations

Loosely speaking, a linear transformation is a mapping from one vector space to another that **preserves** vector space operations.

More precisely, let V, W be vector spaces over the same field F. A **linear transformation** is a mapping $T : V \to W$ such that

$$T(a\boldsymbol{u} + b\boldsymbol{v}) = aT(\boldsymbol{u}) + bT(\boldsymbol{v})$$

for all $a \in F$, $b \in F$ and $\boldsymbol{u} \in V$, $\boldsymbol{v} \in V$.

In this class we will be concerned with linear transformations between (real or complex) **Euclidean spaces**, or subspaces thereof.

Image

$$\text{image}\,(T) = \{\boldsymbol{w} | \, \boldsymbol{w} \in W \ \wedge \ T(\boldsymbol{v}) = \boldsymbol{w} \text{for some} \boldsymbol{v}\}$$

Nullspace

Also known as the kernel:

$$\ker(T) = \{\boldsymbol{v} | \, \boldsymbol{v} \in V \ \wedge \ (T(\boldsymbol{v}) = \boldsymbol{0})\}$$

Both the image and the nullspace are easily seen to be subspaces.

Rank

$$\operatorname{rank}(T) = \dim(\text{image } (T))$$

Nullity

$$\operatorname{null}(T) = \dim(\ker(T))$$

Rank plus nullity theorem

$$\operatorname{rank}(T) + \operatorname{null}(T) = \dim(V)$$

Matrices

Every linear transformation T has a **matrix representation**. If $T : \mathbb{E}^N \to \mathbb{E}^M$, $\mathbb{E} = \mathbb{R}$ or $\mathbb{C}$, then T is represented by an $M \times N$ matrix

$$A = \begin{pmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \dots & a_{MN} \end{pmatrix}$$

where $(a_{1i} \dots a_{Mi})^T = T(e_i)$ and $e_i = (0 \dots 1 \dots 0)^T$ is the i^{th} **standard basis** vector.

Note:A linear transformation can be represented with respect to any bases of $\mathbb{E}^N$ and $\mathbb{E}^M$, leading to a different A. We will always represent a linear transformation using the standard bases.

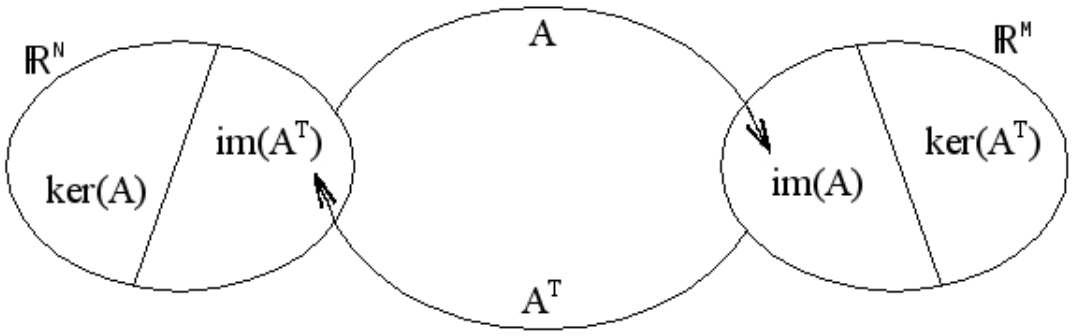
Column span

$$\text{colspan}\,(A) =< A >=\text{image}\,(A)$$

Duality

If $A : \mathbb{R}^N \to \mathbb{R}^M$, then

$$\ker^{\perp}(A) =\text{image}\,\left(A^T\right)$$



If $A : \mathbb{C}^N \to \mathbb{C}^M$, then

$$\ker^{\perp}(A) =\text{image}\,\left(A^{\text{H}}\right)$$

Inverses

The linear transformation/matrix A is **invertible** if and only if there exists a matrix B such that $AB = BA = I$ (identity).

Only **square** matrices can be invertible.

Let $A : \mathbb{F}^N \to \mathbb{F}^N$ be linear, $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$. The following are equivalent:

1. A is invertible (nonsingular)
2. $\text{rank}\,(A) = N$
3. $\text{null}\,(A) = 0$

4. $\det A \neq 0$
5. The columns of A form a basis.

If $A^{-1} = A^T$ (or A^{H} in the complex case), we say A is **orthogonal** (or **unitary**).

Hilbert Spaces
This module will provide an introduction to the concepts of Hilbert spaces.

Hilbert Spaces

A vector space S with a valid inner product defined on it is called an **inner product space**, which is also a **normed linear space**. A **Hilbert space** is an inner product space that is complete with respect to the norm defined using the inner product. Hilbert spaces are named after David Hilbert, who developed this idea through his studies of integral equations. We define our valid norm using the inner product as:
Equation:

$$\| \boldsymbol{x} \| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$$

Hilbert spaces are useful in studying and generalizing the concepts of Fourier expansion, Fourier transforms, and are very important to the study of quantum mechanics. Hilbert spaces are studied under the functional analysis branch of mathematics.

Examples of Hilbert Spaces

Below we will list a few examples of Hilbert spaces. You can verify that these are valid inner products at home.

- For $\mathbb{C}^n$,

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{y}^T \boldsymbol{x} = \begin{pmatrix} y_0 & y_1 & \dots & y_{n-1} \end{pmatrix} \begin{matrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{matrix} = \sum_{i\ 0}^{n-1} x_i y_i$$

- Space of finite energy complex functions: $L^2(\mathbb{R})$

$$\langle \boldsymbol{f}, \boldsymbol{g} \rangle = \int_{-\infty}^{\infty} f(t) g(t) \, \mathrm{d} \, t$$

- Space of square-summable sequences: $\ell^2(\mathbb{Z})$

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i}^{\infty}{}_{-\infty} x[i] y[i]$$

Signal Expansions
The module looks at decomposing signals through orthonormal basis expansion to provide an alternative representation. The module presents many examples of solving these problems and looks at them in several spaces and dimensions.

Main Idea

When working with signals many times it is helpful to break up a signal into smaller, more manageable parts. Hopefully by now you have been exposed to the concept of eigenvectors and there use in decomposing a signal into one of its possible basis. By doing this we are able to simplify our calculations of signals and systems through eigenfunctions of LTI systems.

Now we would like to look at an alternative way to represent signals, through the use of **orthonormal basis**. We can think of orthonormal basis as a set of building blocks we use to construct functions. We will build up the signal/vector as a weighted sum of basis elements.

Example:
The complex sinusoids $\frac{1}{\sqrt{T}}e^{i\omega_0 nt}$ for all $-\infty < n < \infty$ form an orthonormal basis for $L^2([0,T])$.
In our Fourier series equation, $f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\omega_0 nt}$, the $\{c_n\}$ are just another representation of $f(t)$.

Note: For signals/vectors in a Hilbert Space, the expansion coefficients are easy to find.

Alternate Representation

Recall our definition of a **basis**: A set of vectors $\{b_i\}$ in a vector space S is a basis if

1. The b_i are linearly independent.
2. The b_i span S. That is, we can find $\{\alpha_i\}$, where $\alpha_i \in \mathbb{C}$ (scalars) such that
   Equation:

$$\forall x, x \in S : \left(\boldsymbol{x} = \sum_i \alpha_i b_i \right)$$

   where $\boldsymbol{x}$ is a vector in S, α is a scalar in $\mathbb{C}$, and $\boldsymbol{b}$ is a vector in S.

Condition 2 in the above definition says we can **decompose** any vector in terms of the $\{b_i\}$. Condition 1 ensures that the decomposition is **unique** (think about this at home).

Note: The $\{\alpha_i\}$ provide an alternate representation of $\boldsymbol{x}$.

Example:
Let us look at simple example in $\mathbb{R}^2$, where we have the following vector:

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Standard Basis: $\{e_0, e_1\} = \left\{ (10)^T, (01)^T \right\}$

$$\boldsymbol{x} = e_0 + 2e_1$$

Alternate Basis: $\{h_0, h_1\} = \left\{ (11)^T, (1{-}1)^T \right\}$

$$\boldsymbol{x} = \frac{3}{2}h_0 + \frac{-1}{2}h_1$$

In general, given a basis $\{b_0, b_1\}$ and a vector $\boldsymbol{x} \in \mathbb{R}^2$, how do we find the α_0 and α_1 such that
Equation:

$$\boldsymbol{x} = \alpha_0 b_0 + \alpha_1 b_1$$

Finding the Coefficients

Now let us address the question posed above about finding α_i's in general for $\mathbb{R}^2$. We start by rewriting [link] so that we can stack our b_i's as columns in a 2×2 matrix.
Equation:

$$(\boldsymbol{x}) = \alpha_0 (b_0) + \alpha_1 (b_1)$$

Equation:

$$(\boldsymbol{x}) = \begin{pmatrix} \vdots & \vdots \\ b_0 & b_1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$$

Example:
Here is a simple example, which shows a little more detail about the above equations.
Equation:

$$\begin{aligned}\begin{pmatrix} x[0] \\ x[1] \end{pmatrix} &= \alpha_0 \begin{pmatrix} b_0[0] \\ b_0[1] \end{pmatrix} + \alpha_1 \begin{pmatrix} b_1[0] \\ b_1[1] \end{pmatrix} \\ &= \begin{pmatrix} \alpha_0 b_0[0] + \alpha_1 b_1[0] \\ \alpha_0 b_0[1] + \alpha_1 b_1[1] \end{pmatrix}\end{aligned}$$

Equation:

$$\begin{pmatrix} x[0] \\ x[1] \end{pmatrix} = \begin{pmatrix} b_0[0] & b_1[0] \\ b_0[1] & b_1[1] \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$$

Simplifying our Equation

To make notation simpler, we define the following two items from the above equations:

- **Basis Matrix**:

$$B = \begin{pmatrix} \vdots & \vdots \\ b_0 & b_1 \\ \vdots & \vdots \end{pmatrix}$$

- **Coefficient Vector**:

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$$

This gives us the following, concise equation:
Equation:

$$\boldsymbol{x} = B\boldsymbol{\alpha}$$

which is equivalent to $\boldsymbol{x} = \sum_{i=0}^{1} \alpha_i b_i$.

Example:
Given a standard basis, $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$, then we have the following basis matrix:

$$B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

To get the α_i's, we solve for the coefficient vector in [link]
Equation:

$$\boldsymbol{\alpha} = B^{-1}\boldsymbol{x}$$

Where B^{-1} is the inverse matrix of B.

Examples

Example:
Let us look at the standard basis first and try to calculate $\boldsymbol{\alpha}$ from it.

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

Where I is the **identity matrix**. In order to solve for $\boldsymbol{\alpha}$ let us find the inverse of B first (which is obviously very trivial in this case):

$$B^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Therefore we get,

$$\boldsymbol{\alpha} = B^{-1}\boldsymbol{x} = \boldsymbol{x}$$

Example:
Let us look at a ever-so-slightly more complicated basis of $\left\{\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\-1\end{pmatrix}\right\} = \{h_0, h_1\}$ Then our basis matrix and inverse basis matrix becomes:

$$B = \begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$$

$$B^{-1} = \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\\\frac{1}{2} & \frac{-1}{2}\end{pmatrix}$$

and for this example it is given that

$$\boldsymbol{x} = \begin{pmatrix}3\\2\end{pmatrix}$$

Now we solve for $\boldsymbol{\alpha}$

$$\boldsymbol{\alpha} = B^{-1}\boldsymbol{x} = \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\\\frac{1}{2} & \frac{-1}{2}\end{pmatrix}\begin{pmatrix}3\\2\end{pmatrix} = \begin{pmatrix}2.5\\0.5\end{pmatrix}$$

and we get

$$\boldsymbol{x} = 2.5h_0 + 0.5h_1$$

Exercise:

Problem:

Now we are given the following basis matrix and $\boldsymbol{x}$:

$$\{b_0, b_1\} = \left\{ \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix} \right\}$$

$$\boldsymbol{x} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

For this problem, make a sketch of the bases and then represent $\boldsymbol{x}$ in terms of b_0 and b_1.

Solution:

In order to represent $\boldsymbol{x}$ in terms of b_0 and b_1 we will follow the same steps we used in the above example.

$$B = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix}$$

$$B^{-1} = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{-1}{6} \end{pmatrix}$$

$$\boldsymbol{\alpha} = B^{-1}\boldsymbol{x} = \begin{pmatrix} 1 \\ \frac{2}{3} \end{pmatrix}$$

And now we can write $\boldsymbol{x}$ in terms of b_0 and b_1.

$$\boldsymbol{x} = b_0 + \frac{2}{3} b_1$$

And we can easily substitute in our known values of b_0 and b_1 to verify our results.

Note: A change of basis simply looks at $\boldsymbol{x}$ from a "different perspective." B^{-1} **transforms** $\boldsymbol{x}$ from the standard basis to our new basis, $\{b_0, b_1\}$. Notice that this is a totally mechanical procedure.

Extending the Dimension and Space

We can also extend all these ideas past just $\mathbb{R}^2$ and look at them in $\mathbb{R}^n$ and $\mathbb{C}^n$. This procedure extends naturally to higher (> 2) dimensions. Given a basis $\{b_0, b_1, \ldots, b_{n-1}\}$ for $\mathbb{R}^n$, we want to find $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ such that

Equation:

$$\boldsymbol{x} = \alpha_0 b_0 + \alpha_1 b_1 + \ldots + \alpha_{n-1} b_{n-1}$$

Again, we will set up a basis matrix

$$B = \begin{pmatrix} b_0 & b_1 & b_2 & \ldots & b_{n-1} \end{pmatrix}$$

where the columns equal the basis vectors and it will always be an n×n matrix (although the above matrix does not appear to be square since we left terms in vector notation). We can then proceed to rewrite [link]

$$\boldsymbol{x} = \begin{pmatrix} b_0 & b_1 & \ldots & b_{n-1} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{pmatrix} = B\boldsymbol{\alpha}$$

and

$$\boldsymbol{\alpha} = B^{-1}\boldsymbol{x}$$

Introduction to Fourier Analysis
Lists the four Fourier transforms and when to use them.

Fourier's Daring Leap

Fourier postulated around 1807 that any periodic signal (equivalently finite length signal) can be built up as an infinite linear combination of harmonic sinusoidal waves.

i.e. Given the collection
Equation:

$$B = \left\{ e^{j\frac{2\pi}{T}nt} \right\}_{n=-\infty}^{\infty}$$

any
Equation:

$$f(t) \in L^2[0, T)$$

can be approximated arbitrarily closely by
Equation:

$$f(t) = \sum_{n=-\infty}^{\infty} C_n \, e^{j\frac{2\pi}{T}nt}.$$

Now, The issue of exact convergence did bring Fourier much criticism from the French Academy of Science (Laplace, Lagrange, Monge and LaCroix comprised the review committee) for several years after its presentation on 1807. It was not resolved for also a century, and its resolution is interesting and important to understand from a practical viewpoint. See more in the section on **Gibbs Phenomena**.

Fourier analysis is fundamental to understanding the behavior of signals and systems. This is a result of the fact that sinusoids are Eigenfunctions of linear, time-invariant (LTI) systems. This is to say that if we pass any particular sinusoid through a LTI system, we get a scaled version of that same sinusoid on the output. Then, since Fourier analysis allows us to redefine the signals in terms of sinusoids, all we need to do is determine how any given system effects all possible sinusoids (its transfer function) and we have a complete understanding of the system. Furthermore, since we are able to define the passage of sinusoids through a system as multiplication of that sinusoid by the transfer function at the same frequency, we can convert the passage of any signal through a system from convolution (in time) to multiplication (in frequency). These ideas are what give Fourier analysis its power.

Now, after hopefully having sold you on the value of this method of analysis, we must examine exactly what we mean by Fourier analysis. The four Fourier transforms that comprise this analysis are the Fourier Series, Continuous-Time Fourier Transform, Discrete-Time Fourier Transform and Discrete Fourier Transform. For this document, we will view the Laplace Transform and Z-Transform as simply extensions of the CTFT and DTFT respectively. All of these transforms act essentially the same way, by converting a signal in time to an equivalent signal in frequency (sinusoids). However, depending on the nature of a specific signal i.e. whether it is finite- or infinite-length and whether it is discrete- or continuous-time) there is an appropriate transform to convert the signal into the frequency domain. Below is a table of the four Fourier transforms and when each is appropriate. It also includes the relevant convolution for the specified space.

Transform	Time Domain	Frequency Domain	Convolution

Transform	Time Domain	Frequency Domain	Convolution
Continuous-Time Fourier Series	$L^2([0,T))$	$l^2(\mathbb{Z})$	Continuous-Time Circular
Continuous-Time Fourier Transform	$L^2(\mathbb{R})$	$L^2(\mathbb{R})$	Continuous-Time Linear
Discrete-Time Fourier Transform	$l^2(\mathbb{Z})$	$L^2([0,2\pi))$	Discrete-Time Linear
Discrete Fourier Transform	$l^2([0,N-1])$	$l^2([0,N-1])$	Discrete-Time Circular

Table of Fourier Representations

Continuous Time Fourier Transform (CTFT)
Details the Continuous-Time Fourier Transform.

Introduction

In this module, we will derive an expansion for any arbitrary continuous-time function, and in doing so, derive the **Continuous Time Fourier Transform** (CTFT).

Since complex exponentials are eigenfunctions of linear time-invariant (LTI) systems, calculating the output of an LTI system $\mathscr{H}$ given e^{st} as an input amounts to simple multiplication, where $H(s) \in \mathbb{C}$ is the eigenvalue corresponding to s. As shown in the figure, a simple exponential input would yield the output
Equation:

$$y(t) = H(s)e^{st}$$

[missing_resource: simpleLTIsys.png]

Using this and the fact that $\mathscr{H}$ is linear, calculating $y(t)$ for combinations of complex exponentials is also straightforward.

$$c_1 e^{s_1 t} + c_2 e^{s_2 t} \rightarrow c_1 H(s_1) e^{s_1 t} + c_2 H(s_2) e^{s_2 t}$$

$$\sum_n c_n e^{s_n t} \rightarrow \sum_n c_n H(s_n) e^{s_n t}$$

The action of H on an input such as those in the two equations above is easy to explain. $\mathscr{H}$ **independently scales** each exponential component $e^{s_n t}$ by a different complex number $H(s_n) \in \mathbb{C}$. As such, if we can write a function $f(t)$ as a combination of complex exponentials it allows us to easily calculate the output of a system.

Now, we will look to use the power of complex exponentials to see how we may represent arbitrary signals in terms of a set of simpler functions by superposition of a number of complex exponentials. Below we will present

the **Continuous-Time Fourier Transform** (CTFT), commonly referred to as just the Fourier Transform (FT). Because the CTFT deals with nonperiodic signals, we must find a way to include all real frequencies in the general equations. For the CTFT we simply utilize integration over real numbers rather than summation over integers in order to express the aperiodic signals.

Fourier Transform Synthesis

Joseph Fourier demonstrated that an arbitrary $s(t)$ can be written as a linear combination of harmonic complex sinusoids
Equation:

$$s(t) = \sum_{n=-\infty}^{\infty} c_n e^{j\omega_0 nt}$$

where $\omega_0 = \frac{2\pi}{T}$ is the fundamental frequency. For almost all $s(t)$ of practical interest, there exists c_n to make [link] true. If $s(t)$ is finite energy ($s(t) \in L^2[0,T]$), then the equality in [link] holds in the sense of energy convergence; if $s(t)$ is continuous, then [link] holds pointwise. Also, if $s(t)$ meets some mild conditions (the Dirichlet conditions), then [link] holds pointwise everywhere except at points of discontinuity.

The c_n - called the Fourier coefficients - tell us "how much" of the sinusoid $e^{j\omega_0 nt}$ is in $s(t)$. The formula shows $s(t)$ as a sum of complex exponentials, each of which is easily processed by an LTI system (since it is an eigenfunction of **every** LTI system). Mathematically, it tells us that the set of complex exponentials $\left\{\forall n, n \in \mathbb{Z} : \left(e^{j\omega_0 nt}\right)\right\}$ form a basis for the space of T-periodic continuous time functions.

Equations

Now, in order to take this useful tool and apply it to arbitrary non-periodic signals, we will have to delve deeper into the use of the superposition

principle. Let $s_T\left(t\right)$ be a periodic signal having period T. We want to consider what happens to this signal's spectrum as the period goes to infinity. We denote the spectrum for any assumed value of the period by $c_n\left(T\right)$. We calculate the spectrum according to the Fourier formula for a periodic signal, known as the Fourier Series (for more on this derivation, see the section on **Fourier Series**.)
Equation:

$$c_n = \frac{1}{T}\int_0^T s\left(t\right)\,\exp\left(-\imath\omega_0 t\right)\,dt$$

where $\omega_0 = \frac{2\pi}{T}$ and where we have used a symmetric placement of the integration interval about the origin for subsequent derivational convenience. We vary the frequency index n proportionally as we increase the period. Define
Equation:

$$S_T\left(f\right) \equiv Tc_n = \frac{1}{T}\int_0^T \left(S_T\left(f\right)\,\exp\left(\imath\omega_0 t\right)\,dt$$

making the corresponding Fourier Series
Equation:

$$s_T\left(t\right) = \sum_{-\infty}^{\infty} f\left(t\right)\,\exp\left(\imath\omega_0 t\right)\,\frac{1}{T}\right)$$

As the period increases, the spectral lines become closer together, becoming a continuum. Therefore,
Equation:

$$\lim_{T\to\infty}\,s_T\left(t\right) \equiv s\left(t\right) = \int_{-\infty}^{\infty} S\left(f\right)\,\exp\left(\imath\omega_0 t\right)\,df$$

with
Equation:

$$S\left(f\right)=\int_{-\infty}^{\infty}s\left(t\right)\exp\left(-\imath\omega_{0}t\right)dt$$

Equation:

Continuous-Time Fourier Transform

$$\mathscr{F}(\Omega)=\int_{-\infty}^{\infty}f(t)e^{-(i\Omega t)}\ \mathrm{d}\ t$$

Equation:

Inverse CTFT

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}\mathscr{F}(\Omega)e^{i\Omega t}\ \mathrm{d}\ \Omega$$

Note: It is not uncommon to see the above formula written slightly different. One of the most common differences is the way that the exponential is written. The above equations use the radial frequency variable Ω in the exponential, where $\Omega=2\pi f$, but it is also common to include the more explicit expression, $i2\pi ft$, in the exponential. Click here for an overview of the notation used in Connexion's DSP modules.

Example:
We know from Euler's formula that
$\cos\left(\omega t\right)+\sin\left(\omega t\right)=\frac{1-j}{2}e^{j\omega t}+\frac{1+j}{2}e^{-j\omega t}$.

CTFT Definition Demonstration

Interact (when online) with a Mathematica CDF demonstrating Continuous Time Fourier Transform. To Download, right-click and save as .cdf.

Example Problems

Exercise:

Problem: Find the Fourier Transform (CTFT) of the function
Equation:

$$f(t) = \begin{cases} e^{-(\alpha t)} & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Solution:

In order to calculate the Fourier transform, all we need to use is [link], complex exponentials, and basic calculus.
Equation:

$$\begin{aligned}\mathscr{F}(\Omega) &= \int_{-\infty}^{\infty} f(t) e^{-(i\Omega t)} \, \mathrm{d}\, t \\ &= \int_{0}^{\infty} e^{-(\alpha t)} e^{-(i\Omega t)} \, \mathrm{d}\, t \\ &= \int_{0}^{\infty} e^{(-t)(\alpha + i\Omega)} \, \mathrm{d}\, t \\ &= 0 - \tfrac{-1}{\alpha + i\Omega}\end{aligned}$$

Equation:

$$\mathscr{F}(\Omega) = \frac{1}{\alpha + i\Omega}$$

Exercise:

Problem:

Find the inverse Fourier transform of the ideal lowpass filter defined by

Equation:

$$X(\Omega) = \begin{cases} 1 & \text{if } \ |\Omega| \leq \mathrm{M} \\ 0 & \text{otherwise} \end{cases}$$

Solution:

Here we will use [link] to find the inverse FT given that $t \neq 0$.

Equation:

$$\begin{aligned} x(t) &= \tfrac{1}{2\pi} \int_{-M}^{M} e^{i(\Omega, t)} \, \mathrm{d}\, \Omega \\ &= \tfrac{1}{2\pi} e^{i(\Omega, t)} \Big|_{\Omega, \Omega = e^{iw}} \\ &= \tfrac{1}{\pi t} \sin(Mt) \end{aligned}$$

Equation:

$$x(t) = \frac{M}{\pi} \left(\operatorname{sinc} \frac{Mt}{\pi} \right)$$

Fourier Transform Summary

Because complex exponentials are eigenfunctions of LTI systems, it is often useful to represent signals using a set of complex exponentials as a basis. The continuous time Fourier series synthesis formula expresses a continuous time, periodic function as the sum of continuous time, discrete frequency complex exponentials.
Equation:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{j\omega_0 n t}$$

The continuous time Fourier series analysis formula gives the coefficients of the Fourier series expansion.
Equation:

$$c_n = \frac{1}{T} \int_0^T f(t) e^{-(j\omega_0 n t)} \, \mathrm{d}\, t$$

In both of these equations $\omega_0 = \frac{2\pi}{T}$ is the fundamental frequency.

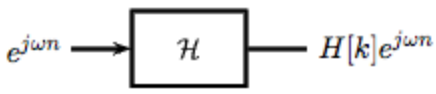
Discrete Time Fourier Transform (DTFT)
Details the discrete-time fourier transform.

Introduction

In this module, we will derive an expansion for arbitrary discrete-time functions, and in doing so, derive the **Discrete Time Fourier Transform** (DTFT).

Since complex exponentials are eigenfunctions of linear time-invariant (LTI) systems, calculating the output of an LTI system $\mathscr{H}$ given $e^{i\omega n}$ as an input amounts to simple multiplication, where $\omega_0 = \frac{2\pi k}{N}$, and where $H[k] \in \mathbb{C}$ is the eigenvalue corresponding to k. As shown in the figure, a simple exponential input would yield the output
Equation:

$$y[n] = H[k]e^{i\omega n}$$



Simple LTI system.

Using this and the fact that $\mathscr{H}$ is linear, calculating $y[n]$ for combinations of **complex exponentials** is also straightforward.

$$c_1 e^{i\omega_1 n} + c_2 e^{i\omega_2 n} \to c_1 H[k_1] e^{i\omega_1 n} + c_2 H[k_2] e^{i\omega_1 n}$$

$$\sum_l c_l e^{i\omega_l n} \to \sum_l c_l H[k_l] e^{i\omega_l n}$$

The action of H on an input such as those in the two equations above is easy to explain. $\mathscr{H}$ **independently scales** each exponential component $e^{i\omega_1 n}$ by a different complex number $H[k_l] \in \mathbb{C}$. As such, if we can write a function $y[n]$ as a combination of complex exponentials it allows us to easily calculate the output of a system.

Now, we will look to use the power of complex exponentials to see how we may represent arbitrary signals in terms of a set of simpler functions by superposition of a number of complex exponentials. Below we will present the **Discrete-Time Fourier Transform** (DTFT). Because the DTFT deals with nonperiodic signals, we must find a way to include all real frequencies in the general equations. For the DTFT we simply utilize summation over all real numbers rather than summation over integers in order to express the aperiodic signals.

DTFT synthesis

It can be demonstrated that an arbitrary Discrete Time-periodic function $f[n]$ can be written as a linear combination of harmonic complex sinusoids
Equation:

$$f[n] = \sum_{k=0}^{N-1} c_k e^{i\omega_0 kn}$$

where $\omega_0 = \frac{2\pi}{N}$ is the fundamental frequency. For almost all $f[n]$ of practical interest, there exists c_n to make [link] true. If $f[n]$ is finite energy ($f[n] \in L^2[0, N]$), then the equality in [link] holds in the sense of energy convergence; with discrete-time signals, there are no concerns for divergence as there are with continuous-time signals.

The c_n - called the Fourier coefficients - tell us "how much" of the sinusoid $e^{j\omega_0 kn}$ is in $f[n]$. The formula shows $f[n]$ as a sum of complex exponentials, each of which is easily processed by an LTI system (since it is an eigenfunction of **every** LTI system). Mathematically, it tells us that the

set of complex exponentials $\left\{\forall k, k \in \mathbb{Z} : \left(e^{j\omega_0 kn}\right)\right\}$ form a basis for the space of N-periodic discrete time functions.

Equations

Now, in order to take this useful tool and apply it to arbitrary non-periodic signals, we will have to delve deeper into the use of the superposition principle. Let $s_T\left(t\right)$ be a periodic signal having period T. We want to consider what happens to this signal's spectrum as the period goes to infinity. We denote the spectrum for any assumed value of the period by $c_n\left(T\right)$. We calculate the spectrum according to the Fourier formula for a periodic signal, known as the Fourier Series (for more on this derivation, see the section on **Fourier Series**.)
Equation:

$$c_n = \frac{1}{T}\int_0^T s\left(t\right)\,\exp\left(-\imath\omega_0 t\right)\,dt$$

where $\omega_0 = \frac{2\pi}{T}$ and where we have used a symmetric placement of the integration interval about the origin for subsequent derivational convenience. We vary the frequency index n proportionally as we increase the period. Define
Equation:

$$S_T\left(f\right) \equiv Tc_n = \frac{1}{T}\int_0^T \left(S_T\left(f\right)\,\exp\left(\imath\omega_0 t\right)\,dt\right.$$

making the corresponding Fourier Series
Equation:

$$s_T\left(t\right) = \sum_{-\infty}^{\infty} f\left(t\right)\,\exp\left(\imath\omega_0 t\right)\,\frac{1}{T}\right)$$

As the period increases, the spectral lines become closer together, becoming a continuum. Therefore,

Equation:

$$\lim_{T \to \infty} s_T(t) \equiv s(t) = \int_{-\infty}^{\infty} S(f) \exp(\imath \omega_0 t) \, df$$

with

Equation:

$$S(f) = \int_{-\infty}^{\infty} s(t) \exp(-\imath \omega_0 t) \, dt$$

Equation:

Discrete-Time Fourier Transform

$$\mathscr{F}(\omega) = \sum_{n=-\infty}^{\infty} f[n] e^{-(i \omega n)}$$

Equation:

Inverse DTFT

$$f[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathscr{F}(\omega) e^{i \omega n} \, \mathrm{d}\, \omega$$

Note: It is not uncommon to see the above formula written slightly different. One of the most common differences is the way that the exponential is written. The above equations use the radial frequency variable ω in the exponential, where $\omega = 2\pi f$, but it is also common to include the more explicit expression, $i 2 \pi f t$, in the exponential. Sometimes DTFT notation is expressed as $F\left(e^{i\omega}\right)$, to make it clear that it is not a

CTFT (which is denoted as $F(\Omega)$). Click here for an overview of the notation used in Connexion's DSP modules.

DTFT Definition demonstration

Click on the above thumbnail image (when online) to download an interactive Mathematica Player demonstrating Discrete Time Fourier Transform. To Download, right-click and save target as .cdf.

DTFT Summary

Because complex exponentials are eigenfunctions of LTI systems, it is often useful to represent signals using a set of complex exponentials as a basis. The discrete time Fourier transform synthesis formula expresses a discrete time, aperiodic function as the infinite sum of continuous frequency complex exponentials.

Equation:

$$\mathscr{F}(\omega) = \sum_{n=-\infty}^{\infty} f[n] e^{-(i\omega n)}$$

The discrete time Fourier transform analysis formula takes the same discrete time domain signal and represents the signal in the continuous frequency domain.

Equation:

$$f[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathscr{F}(\omega) e^{i\omega n} \, \mathrm{d}\, \omega$$

DFT as a Matrix Operation

Matrix Review

Recall:

- Vectors in $\mathbb{R}^N$:

$$\forall x_i, x_i \in \mathbb{R}: \quad \boldsymbol{x} = \begin{matrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \end{matrix}$$

- Vectors in $\mathbb{C}^N$:

$$\forall x_i, x_i \in \mathbb{C}: \quad \boldsymbol{x} = \begin{matrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \end{matrix}$$

- Transposition:

    1. transpose:

$$\boldsymbol{x}^T = \begin{pmatrix} x_0 & x_1 & \dots & x_{N-1} \end{pmatrix}$$

    2. conjugate:

$$\boldsymbol{x}^{\mathrm{H}} = \begin{pmatrix} x_0 & x_1 & \dots & x_{N-1} \end{pmatrix}$$

- Inner product:

    1. real:

$$\boldsymbol{x}^T \boldsymbol{y} = \sum_{i=0}^{N-1} x_i y_i$$

2. complex:

$$\boldsymbol{x}^{\mathrm{H}}\boldsymbol{y} = \sum_{i=0}^{N-1} x_n y_n$$

- Matrix Multiplication:

$$A\boldsymbol{x} = \begin{matrix} a_{00} & a_{01} & \dots & a_{0,N-1} \\ a_{10} & a_{11} & \dots & a_{1,N-1} \\ \vdots & \vdots & \dots & \vdots \\ a_{N-1,0} & a_{N-1,1} & \dots & a_{N-1,N-1} \end{matrix} \begin{matrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \end{matrix} = \begin{matrix} y_0 \\ y_1 \\ \dots \\ y_{N-1} \end{matrix}$$

$$y_k = \sum_{n=0}^{N-1} a_{kn} x_n$$

- Matrix Transposition:

$$A^T = \begin{matrix} a_{00} & a_{10} & \dots & a_{N-1,0} \\ a_{01} & a_{11} & \dots & a_{N-1,1} \\ \vdots & \vdots & \dots & \vdots \\ a_{0,N-1} & a_{1,N-1} & \dots & a_{N-1,N-1} \end{matrix}$$

Matrix transposition involved simply swapping the rows with columns.

$$A^{\mathrm{H}} = A^T$$

The above equation is Hermitian transpose.

$$\boldsymbol{A}^T{}_{k,n} = \boldsymbol{A}_{n,k}$$

$$A^{\mathrm{H}}{}_{k,n} = \boldsymbol{A}_{\,n,k}$$

Representing DFT as Matrix Operation

Now let's represent the DFT in vector-matrix notation.

$$\boldsymbol{x} = \begin{matrix} x[0] \\ x[1] \\ \cdots \\ x[N-1] \end{matrix}$$

$$\boldsymbol{X} = \begin{matrix} X[0] \\ X[1] \\ \cdots \\ X[N-1] \end{matrix} \quad \in \mathbb{C}^N$$

Here $\boldsymbol{x}$ is the vector of time samples and $\boldsymbol{X}$ is the vector of DFT coefficients. How are $\boldsymbol{x}$ and $\boldsymbol{X}$ related:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-(i\frac{2\pi}{N}kn)}$$

where

$$a_{kn} = \left(e^{-(i\frac{2\pi}{N})}\right)^{kn} = {W_N}^{kn}$$

so

$$\boldsymbol{X} = W\boldsymbol{x}$$

where $\boldsymbol{X}$ is the DFT vector, W is the matrix and $\boldsymbol{x}$ the time domain vector.

$$W_{k,n} = \left(e^{-(i\frac{2\pi}{N})}\right)^{kn}$$

$$\boldsymbol{X} = W \begin{matrix} x[0] \\ x[1] \\ \cdots \\ x[N-1] \end{matrix}$$

IDFT:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \left(e^{i \frac{2\pi}{N}} \right)^{nk}$$

where

$$\left(e^{i \frac{2\pi}{N}} \right)^{nk} = {W_N}^{nk}$$

${W_N}^{nk}$ is the matrix Hermitian transpose. So,

$$\boldsymbol{x} = \frac{1}{N} W^{\mathrm{H}} \boldsymbol{X}$$

where $\boldsymbol{x}$ is the time vector, $\frac{1}{N} W^{\mathrm{H}}$ is the inverse DFT matrix, and $\boldsymbol{X}$ is the DFT vector.

The FFT Algorithm

FFT
: (Fast Fourier Transform) An efficient **computational algorithm** for computing the DFT.

The Fast Fourier Transform FFT

DFT can be **expensive** to compute **directly**

$$\forall k, 0 \leq k \leq N-1 : \left(X[k] = \sum_{n=0}^{N-1} x[n] e^{-(i2\pi \frac{k}{N} n)} \right)$$

For each k, we must execute:

- N complex multiplies
- $N-1$ complex adds

The total cost of direct computation of an N-point DFT is

- N^2 complex multiplies
- $N\left(N-1\right)$ complex adds

How many adds and mults of **real** numbers are required?

This " $O\left(N^2\right)$" computation rapidly gets out of hand, as N gets large:

N	1	10	100	1000	10^6
N^2	1	100	10,000	10^6	10^{12}

Image not finished

The FFT provides us with a much more **efficient** way of computing the DFT. The FFT requires only " $O(N \log N)$" computations to compute the N-point DFT.

N	10	100	1000	10^6
N^2	100	10,000	10^6	10^{12}
$N \log N$	10	200	3000	6×10^6

How long is 10^{12}μsec? More than 10 days! How long is 6×10^6μsec?

Image not finished

The FFT and digital computers revolutionized DSP (1960 - 1980).

How does the FFT work?

- The FFT exploits the **symmetries** of the complex exponentials ${W_N}^{kn} = e^{-\left(i\frac{2\pi}{N}kn\right)}$
- ${W_N}^{kn}$ are called "**twiddle factors**"

Symmetry
Complex Conjugate Symmetry

$$ {W_N}^{k(N-n)} = {W_N}^{-(kn)} = \overline{{W_N}^{kn}} $$

$$ e^{-\left(i2\pi\frac{k}{N}(N-n)\right)} = e^{i2\pi\frac{k}{N}n} = \overline{e^{-\left(i2\pi\frac{k}{N}n\right)}} $$

Symmetry
Periodicity in n and k

$$W_N{}^{kn} = W_N{}^{k(N+n)} = W_N{}^{(k+N)n}$$

$$e^{-\left(i\frac{2\pi}{N}kn\right)} = e^{-\left(i\frac{2\pi}{N}k(N+n)\right)} = e^{-\left(i\frac{2\pi}{N}(k+N)n\right)}$$

$$W_N = e^{-\left(i\frac{2\pi}{N}\right)}$$

Decimation in Time FFT

- Just one of **many** different FFT algorithms
- The **idea** is to build a DFT out of smaller and smaller DFTs by decomposing $x[n]$ into smaller and smaller subsequences.
- Assume $N = 2^m$ (a power of 2)

Derivation

N is **even**, so we can complete $X[k]$ by separating $x[n]$ into **two** subsequences each of length $\frac{N}{2}$.

$$x[n] \rightarrow \begin{cases} \frac{N}{2} & \text{if } \ n = \text{even} \\ \frac{N}{2} & \text{if } \ n = \text{odd} \end{cases}$$

$$\forall k, 0 \leq k \leq N-1 : \left(X[k] = \sum_{n=0}^{N-1} x[n] W_N{}^{kn} \right)$$

$$X[k] = \sum_{n=2r} x[n] W_N{}^{kn} + \sum_{n=2r+1} x[n] W_N{}^{kn}$$

where $0 \leq r \leq \frac{N}{2} - 1$. So
Equation:

$$\begin{aligned} X[k] &= \sum_{r=0}^{\frac{N}{2}-1} x[2r] {W_N}^{2kr} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] {W_N}^{(2r+1)k} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x[2r] \left({W_N}^2\right)^{kr} + {W_N}^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] \left({W_N}^2\right)^{kr} \end{aligned}$$

where ${W_N}^2 = e^{-\left(i\frac{2\pi}{N}2\right)} = e^{-\left(i\frac{2\pi}{\frac{N}{2}}\right)} = W_{\frac{N}{2}}$. So

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] {W_{\frac{N}{2}}}^{kr} + {W_N}^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] {W_{\frac{N}{2}}}^{kr}$$

where $\sum_{r=0}^{\frac{N}{2}-1} x[2r] {W_{\frac{N}{2}}}^{kr}$ is $\frac{N}{2}$-point DFT of even samples ($G[k]$) and $\sum_{r=0}^{\frac{N}{2}-1} x[2r+1] {W_{\frac{N}{2}}}^{kr}$ is $\frac{N}{2}$-point DFT of odd samples ($H[k]$).

$$\forall k, 0 \leq k \leq N-1 : \left(X[k] = G[k] + {W_N}^k H[k]\right)$$

Decomposition of an N-point DFT as a sum of 2 $\frac{N}{2}$-point DFTs.

Why would we want to do this? **Because it is more efficient!**

Note: Cost to compute an N-point DFT is approximately N^2 complex mults and adds.

But decomposition into 2 $\frac{N}{2}$-point DFTs + combination requires only

$$\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N$$

where the first part is the number of complex mults and adds for $\frac{N}{2}$-point DFT, $G[k]$. The second part is the number of complex mults and adds for $\frac{N}{2}$

-point DFT, $H[k]$. The third part is the number of complex mults and adds for combination. And the total is $\frac{N^2}{2} + N$ complex mults and adds.

Example:
Savings
For $N = 1000$,

$$N^2 = 10^6$$

$$\frac{N^2}{2} + N = \frac{10^6}{2} + 1000$$

Because 1000 is small compared to 500,000,

$$\frac{N^2}{2} + N \simeq \frac{10^6}{2}$$

So why stop here?! Keep decomposing. Break each of the $\frac{N}{2}$-point DFTs into two $\frac{N}{4}$-point DFTs, etc., ...

We can keep decomposing:

$$\frac{N}{2^1} = \left\{ \frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \ldots, \frac{N}{2^{m-1}}, \frac{N}{2^m} \right\} = 1$$

where

$$m = \log_2 N = \text{ times}$$

Computational cost: N-pt DFT two $\frac{N}{2}$-pt DFTs. The cost is $N^2 \to 2\left(\frac{N}{2}\right)^2 + N$. So replacing each $\frac{N}{2}$-pt DFT with two $\frac{N}{4}$-pt DFTs will reduce cost to

$$2\left(2\left(\frac{N}{4}\right)^2+\frac{N}{2}\right)+N=4\left(\frac{N}{4}\right)^2+2N=\frac{N^2}{2^2}+2N=\frac{N^2}{2^p}+pN$$

As we keep going $p=\{3,4,\ldots,m\}$, where $m=\log_2 N$. We get the cost

$$\frac{N^2}{2^{\log_2 N}}+N\log_2 N=\frac{N^2}{N}+N\log_2 N=N+N\log_2 N$$

$N+N\log_2 N$ is the total number of complex adds and mults.

For large N, cost $\simeq N\log_2 N$ or " $O(N\log_2 N)$", since $N\log_2 N \gg N$ for large N.

Image not finished

$N=8$ point FFT.
Summing nodes
${W_n}^k$ twiddle
multiplication
factors.

Note: Weird order of time samples

Image not finished

This is called
"butterflies."

Introduction

Contents of Sampling chapter

Why sample?

This section introduces sampling. Sampling is the necessary fundament for all digital signal processing and communication. Sampling can be defined as the process of measuring an analog signal at distinct points.

Digital representation of analog signals offers advantages in terms of

- robustness towards noise, meaning we can send more bits/s
- use of flexible processing equipment, in particular the computer
- more reliable processing equipment
- easier to adapt complex algorithms

Claude E. Shannon

Claude
Elwood
Shannon
(1916-2001)

Claude Shannon has been called the father of information theory, mainly due to his landmark papers on the "Mathematical theory of communication". Harry Nyquist was the first to state the sampling theorem in 1928, but it was not proven until Shannon proved it 21 years later in the paper "Communications in the presence of noise".

Notation

In this chapter we will be using the following notation

- Original analog signal $x(t)$
- Sampling frequency F_s
- Sampling interval T_s (Note that: $F_s = \frac{1}{T_s}$)
- Sampled signal $x_s(n)$. (Note that $x_s(n) = x(nT_s)$)
- Real angular frequency Ω
- Digital angular frequency ω. (Note that: $\omega = \Omega T_s$)

The Sampling Theorem

Note: When sampling an analog signal the sampling frequency must be greater than twice the highest frequency component of the analog signal to be able to reconstruct the original signal from the sampled version.

Finished? Have at look at: Proof Illustrations Matlab Example Aliasing applet Hold operation System view Exercises

Proof

Note: In order to recover the signal $x(t)$ from it's samples exactly, it is necessary to sample $x(t)$ at a rate greater than twice it's highest frequency component.

Introduction

As mentioned earlier, sampling is the necessary fundament when we want to apply digital signal processing on analog signals.

Here we present the proof of the sampling theorem. The proof is divided in two. First we find an expression for the spectrum of the signal resulting from sampling the original signal $x(t)$. Next we show that the signal $x(t)$ can be recovered from the samples. Often it is easier using the frequency domain when carrying out a proof, and this is also the case here.

Key points in the proof

- We find an equation for the spectrum of the sampled signal
- We find a simple method to reconstruct the original signal
- The sampled signal has a periodic spectrum...
- ...and the period is $2 \times \pi F_s$

Proof part 1 - Spectral considerations

By sampling $x(t)$ every T_s second we obtain $x_s(n)$. The inverse fourier transform of this time discrete signal is

Equation:

$$x_s(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X_s\left(e^{i\omega}\right) e^{i\omega n} \, \mathrm{d}\, \omega$$

For convenience we express the equation in terms of the real angular frequency Ω using $\omega = \Omega T_s$. We then obtain
Equation:

$$x_s(n) = \frac{T_s}{2\pi} \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} X_s\left(e^{i\Omega T_s}\right) e^{i\Omega T_s n} \, \mathrm{d}\, \Omega$$

The inverse fourier transform of a continuous signal is
Equation:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(i\Omega) e^{i\Omega t} \, \mathrm{d}\, \Omega$$

From this equation we find an expression for $x\left(nT_s\right)$
Equation:

$$x(nT_s) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(i\Omega) e^{i\Omega n T_s} \, \mathrm{d}\, \Omega$$

To account for the difference in region of integration we split the integration in [link] into subintervals of length $\frac{2\pi}{T_s}$ and then take the sum over the resulting integrals to obtain the complete area.
Equation:

$$x(nT_s) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \int_{\frac{(2k-1)\pi}{T_s}}^{\frac{(2k+1)\pi}{T_s}} X(i\Omega) e^{i\Omega n T_s} \, \mathrm{d}\, \Omega$$

Then we change the integration variable, setting $\Omega = \eta + \frac{2 \times \pi k}{T_s}$
Equation:

$$x(nT_s) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} X\left(i\left(\eta + \frac{2 \times \pi k}{T_s}\right)\right) e^{i\left(\eta + \frac{2 \times \pi k}{T_s}\right) n T_s} \, \mathrm{d}\, \eta$$

We obtain the final form by observing that $e^{i2 \times \pi k n} = 1$, reinserting $\eta = \Omega$ and multiplying by $\frac{T_s}{T_s}$

Equation:

$$x(nT_s) = \frac{T_s}{2\pi} \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} \sum_{k=-\infty}^{\infty} \frac{1}{T_s} X\left(i\left(\Omega + \frac{2 \times \pi k}{T_s}\right)\right) e^{i \Omega n T_s} \, \mathrm{d}\, \Omega$$

To make $x_s(n) = x(nT_s)$ for all values of n, the integrands in [link] and [link] have to agreee, that is

Equation:

$$X_s\left(e^{i \Omega T_s}\right) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X\left(i\left(\Omega + \frac{2\pi k}{T_s}\right)\right)$$

This is a central result. We see that the digital spectrum consists of a sum of shifted versions of the original, analog spectrum. Observe the periodicity!

We can also express this relation in terms of the digital angular frequency $\omega = \Omega T_s$

Equation:

$$X_s\left(e^{i \omega}\right) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X\left(i\frac{\omega + 2 \times \pi k}{T_s}\right)$$

This concludes the first part of the proof. Now we want to find a reconstruction formula, so that we can recover $x(t)$ from $x_s(n)$.

Proof part II - Signal reconstruction

For a bandlimited signal the inverse fourier transform is
Equation:

$$x(t) = \frac{1}{2\pi} \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} X(i\Omega) e^{i\Omega t} \, \mathrm{d}\, \Omega$$

In the interval we are integrating we have: $X_s\left(e^{i\Omega T_s}\right) = \frac{X(i\Omega)}{T_s}$.
Substituting this relation into [link] we get
Equation:

$$x(t) = \frac{T_s}{2\pi} \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} X_s\left(e^{i\Omega T_s}\right) e^{i\Omega t} \, \mathrm{d}\, \Omega$$

Using the DTFT relation for $X_s\left(e^{i\Omega T_s}\right)$ we have
Equation:

$$x(t) = \frac{T_s}{2\pi} \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} \sum_{n=-\infty}^{\infty} x_s(n) e^{-(i\Omega n T_s)} e^{i\Omega t} \, \mathrm{d}\, \Omega$$

Interchanging integration and summation (under the assumption of convergence) leads to
Equation:

$$x(t) = \frac{T_s}{2\pi} \sum_{n=-\infty}^{\infty} x_s(n) \int_{\frac{-\pi}{T_s}}^{\frac{\pi}{T_s}} e^{i\Omega(t - nT_s)} \, \mathrm{d}\, \Omega$$

Finally we perform the integration and arrive at the important reconstruction formula
Equation:

$$x(t) = \sum_{n=-\infty}^{\infty} x_s(n) \frac{\sin\left(\frac{\pi}{T_s}\left(t - nT_s\right)\right)}{\frac{\pi}{T_s}\left(t - nT_s\right)}$$

(Thanks to R.Loos for pointing out an error in the proof.)

Summary

Note: $X_s\left(e^{i\Omega T_s}\right) = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} X\left(i\left(\Omega + \frac{2\pi k}{T_s}\right)\right)$

Note: $x(t) = \sum_{n=-\infty}^{\infty} x_s(n) \frac{\sin\left(\frac{\pi}{T_s}(t-nT_s)\right)}{\frac{\pi}{T_s}(t-nT_s)}$

Go to Introduction Illustrations Matlab Example Hold operation Aliasing applet System view Exercises ?

Illustrations

In this module we illustrate the processes involved in sampling and reconstruction. To see how all these processes work together as a whole, take a look at the system view. In Sampling and reconstruction with Matlab we provide a Matlab script for download. The matlab script shows the process of sampling and reconstruction **live**.

Basic examples

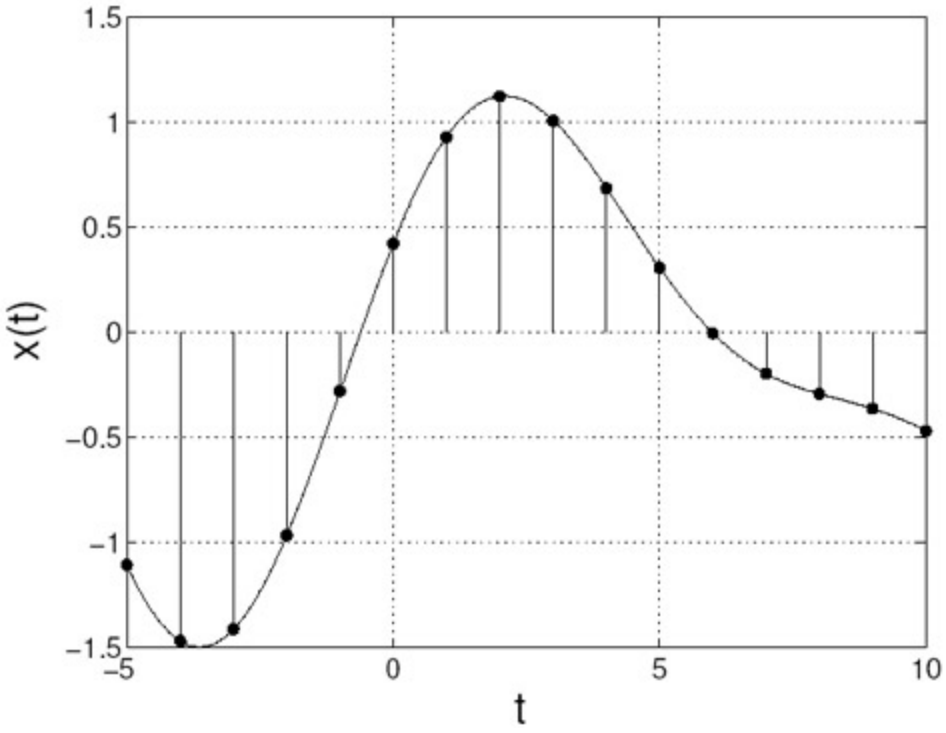
Example:
To sample an analog signal with 3000 Hz as the highest frequency component requires sampling at 6000 Hz or above.

Example:
The sampling theorem can also be applied in two dimensions, i.e. for image analysis. A 2D sampling theorem has a simple physical interpretation in image analysis: Choose the sampling interval such that it is less than or equal to half of the smallest interesting detail in the image.

The process of sampling

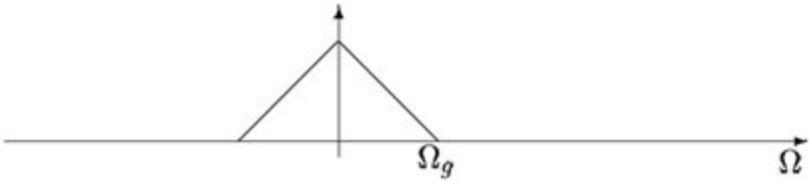
We start off with an analog signal. This can for example be the sound coming from your stereo at home or your friend talking.

The signal is then sampled uniformly. Uniform sampling implies that we sample every T_s seconds. In [link] we see an analog signal. The analog signal has been sampled at times $t = nT_s$.



Analog signal, samples are marked with dots.

In signal processing it is often more convenient and easier to work in the frequency domain. So let's look at at the signal in frequency domain, [link]. For illustration purposes we take the frequency content of the signal as a triangle. (If you Fourier transform the signal in [link] you will not get such a nice triangle.)



The spectrum $X(i\Omega)$.

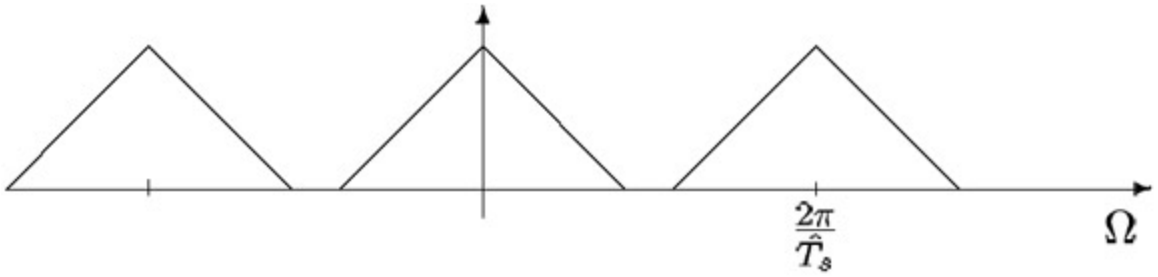
Notice that the signal in [link] is bandlimited. We can see that the signal is bandlimited because $X(i\Omega)$ is zero outside the interval $[-\Omega_g, \Omega_g]$. Equivalentely we can state that the signal has no angular frequencies above Ω_g, corresponding to no frequencies above $F_g = \frac{\Omega_g}{2\pi}$.

Now let's take a look at the sampled signal in the frequency domain. While proving the sampling theorem we found the the spectrum of the sampled signal consists of a sum of shifted versions of the analog spectrum. Mathematically this is described by the following equation:
Equation:

$$X_s\left(e^{i\Omega T_s}\right) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X\left(i\left(\Omega + \frac{2\pi k}{T_s}\right)\right)$$

Sampling fast enough

In [link] we show the result of sampling $x(t)$ according to the sampling theorem. This means that when sampling the signal in [link]/[link] we use $F_s \geq 2F_g$. Observe in [link] that we have the same spectrum as in [link] for $\Omega \in$ [-Ω_g, Ω_g], except for the scaling factor $\frac{1}{T_s}$. This is a consequence of the sampling frequency. As mentioned in the proof the spectrum of the sampled signal is periodic with period $2\pi F_s = \frac{2\pi}{T_s}$.



The spectrum X_s. Sampling frequency is OK.

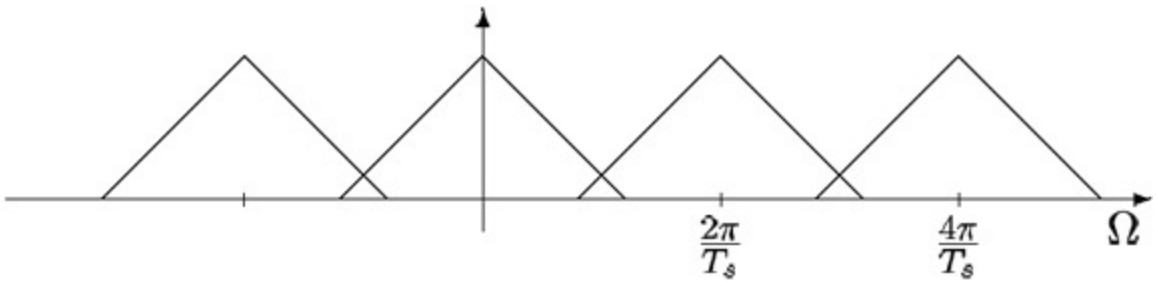
So now we are, according to the sample theorem, able to reconstruct the original signal **exactly**. How we can do this will be explored further down under reconstruction. But first we will take a look at what happens when we sample too slowly.

Sampling too slowly

If we sample $x(t)$ too slowly, that is $F_s < 2F_g$, we will get overlap between the repeated spectra, see [link]. According to [link] the resulting spectra is the sum of these. This overlap gives rise to the concept of aliasing.

> **Note:** If the sampling frequency is less than twice the highest frequency component, then frequencies in the original signal that are above half the sampling rate will be "aliased" and will appear in the resulting signal as lower frequencies.

The consequence of aliasing is that we cannot recover the original signal, so aliasing has to be avoided. Sampling too slowly will produce a sequence $x_s(n)$ that could have orginated from a number of signals. So there is **no** chance of recovering the original signal. To learn more about aliasing, take a look at this module. (Includes an applet for demonstration!)



The spectrum X_s. Sampling frequency is too low.

To avoid aliasing we have to sample fast enough. But if we can't sample fast enough (possibly due to costs) we can include an Anti-Aliasing filter. This will not able us to get an exact reconstruction but can still be a good solution.

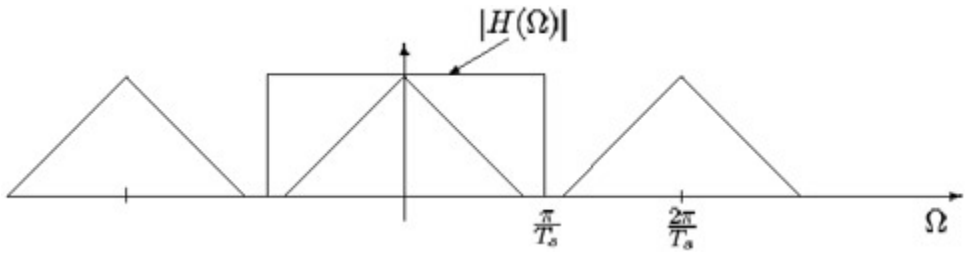
Note: Typically a low-pass filter that is applied before sampling to ensure that no components with frequencies greater than half the sample frequency remain.

Example:
The stagecoach effect
In older western movies you can observe aliasing on a stagecoach when it starts to roll. At first the spokes appear to turn forward, but as the stagecoach increase its speed the spokes appear to turn backward. This comes from the fact that the sampling rate, here the number of frames per second, is too low. We can view each frame as a sample of an image that is changing continuously in time. (Applet illustrating the stagecoach effect)

Reconstruction

Given the signal in [link] we want to recover the original signal, but the question is how?

When there is no overlapping in the spectrum, the spectral component given by $k = 0$ (see [link]),is equal to the spectrum of the analog signal. This offers an oppurtunity to use a simple reconstruction process. Remember what you have learned about filtering. What we want is to change signal in [link] into that of [link]. To achieve this we have to remove all the extra components generated in the sampling process. To remove the extra components we apply an ideal analog low-pass filter as shown in [link] As we see the ideal filter is rectangular in the frequency domain. A rectangle in the frequency domain corresponds to a sinc function in time domain (and vice versa).



$H(i\Omega)$ The ideal reconstruction filter.

Then we have reconstructed the original spectrum, and as we know **if two signals are identical in the frequency domain, they are also identical in the time domain**. End of reconstruction.

Conclusions

The Shannon sampling theorem requires that the input signal prior to sampling is band-limited to at most half the sampling frequency. Under this condition the samples give an exact signal representation. It is truly remarkable that such a broad and useful class signals can be represented that easily!

We also looked into the problem of reconstructing the signals form its samples. Again the simplicity of the **principle** is striking: linear filtering by an ideal low-pass filter will do the job. However, the ideal filter is impossible to create, but that is another story...

Go to? Introduction Proof Illustrations Matlab Example Aliasing applet Hold operation System view Exercises

Sampling and reconstruction with Matlab

Matlab files

Samprecon.m

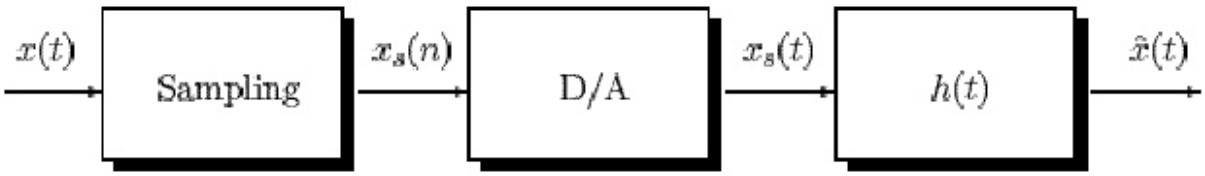
Introduction Proof Illustrations Aliasing applet Hold operation System view Exercises

Systems view of sampling and reconstruction

Ideal reconstruction system

[link] shows the ideal reconstruction system based on the results of the Sampling theorem proof.

[link] consists of a sampling device which produces a time-discrete sequence $x_s(n)$. The reconstruction filter, $h(t)$, is an ideal analog sinc filter, with $h(t) = \text{sinc}\left(\frac{t}{T_s}\right)$. We can't apply the time-discrete sequence $x_s(n)$ directly to the analog filter $h(t)$. To solve this problem we turn the sequence into an analog signal using delta functions. Thus we write $x_s(t) = \sum_{n=-\infty}^{\infty} x_s(n)\delta(t - nT)$.

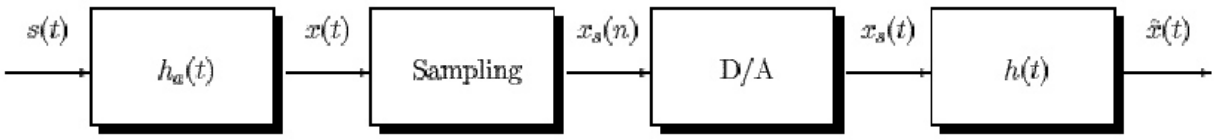


Ideal reconstruction system

But when will the system produce an output $\widehat{x}(t) = x(t)$? According to the sampling theorem we have $\widehat{x}(t) = x(t)$ when the sampling frequency, F_s, is at least twice the highest frequency component of $x(t)$.

Ideal system including anti-aliasing

To be sure that the reconstructed signal is free of aliasing it is customary to apply a lowpass filter, an anti-aliasing filter, before sampling as shown in [link].



Ideal reconstruction system with anti-aliasing filter

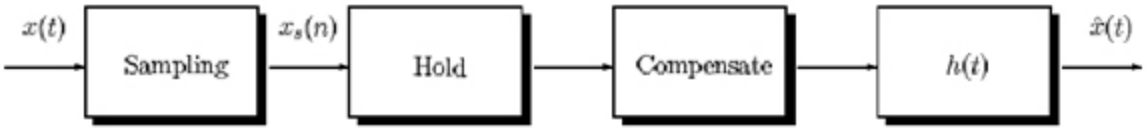
Again we ask the question of when the system will produce an output $\widehat{x}(t) = s(t)$? If the signal is entirely confined within the passband of the lowpass filter we will get perfect reconstruction if F_s is high enough.

But if the anti-aliasing filter removes the "higher" frequencies, (which in fact is the job of the anti-aliasing filter), we will **never** be able to **exactly** reconstruct the original signal, $s(t)$. If we sample fast enough we can reconstruct $x(t)$, which in most cases is satisfying.

The reconstructed signal, $\widehat{x}(t)$, will not have aliased frequencies. This is essential for further use of the signal.

Reconstruction with hold operation

To make our reconstruction system realizable there are many things to look into. Among them are the fact that any practical reconstruction system must input finite length pulses into the reconstruction filter. This can be accomplished by the hold operation. To alleviate the distortion caused by the hold opeator we apply the output from the hold device to a compensator. The compensation can be as accurate as we wish, this is cost and application consideration.



More practical reconstruction system with a hold component

By the use of the hold component the reconstruction will not be exact, but as mentioned above we can get as close as we want.

Introduction Proof Illustrations Matlab example Hold operation Aliasing applet Exercises

Sampling CT Signals: A Frequency Domain Perspective

Understanding Sampling in the Frequency Domain

We want to relate $x_c(t)$ directly to $x[n]$. Compute the CTFT of

$$x_s(t) = \sum_{n=-\infty}^{\infty} x_c(nT)\delta(t - nT)$$

Equation:

$$\begin{aligned} X_s(\Omega) &= \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x_c(nT)\delta(t - nT)e^{(-i)\Omega t}\ \mathrm{d}\ t \\ &= \sum_{n=-\infty}^{\infty} x_c(nT) \int_{-\infty}^{\infty} \delta(t - nT)e^{(-i)\Omega t}\ \mathrm{d}\ t \\ &= \sum_{n=-\infty}^{\infty} x[n]e^{(-i)\Omega nT} \\ &= \sum_{n=-\infty}^{\infty} x[n]e^{(-i)\omega n} \\ &= X(\omega) \end{aligned}$$

where $\omega \equiv \Omega T$ and $X(\omega)$ is the DTFT of $x[n]$.

Note:

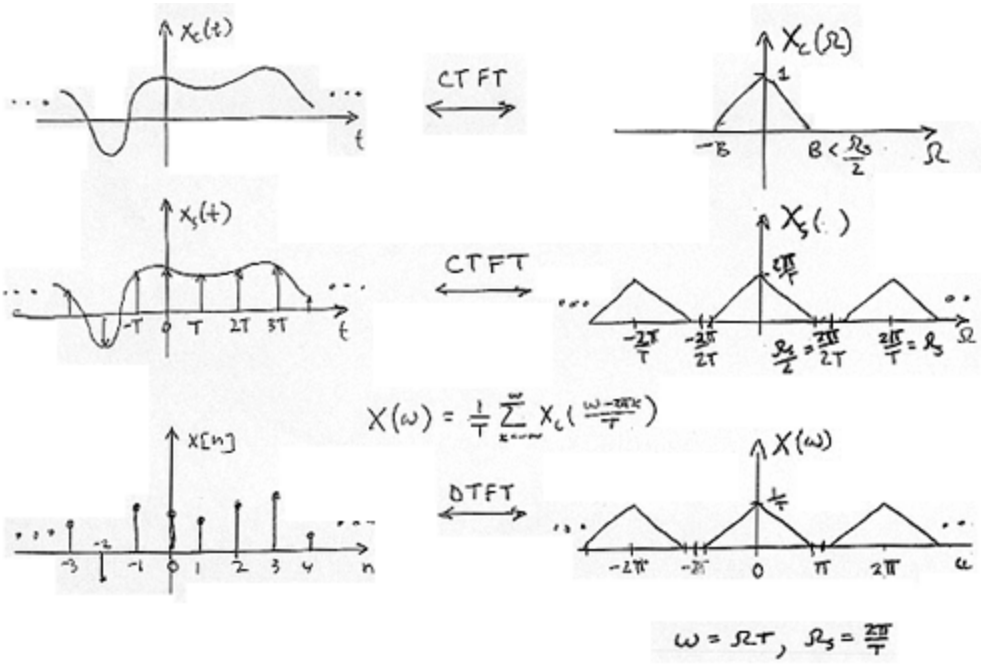
$$X_s(\Omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c(\Omega - k\Omega_s)$$

Equation:

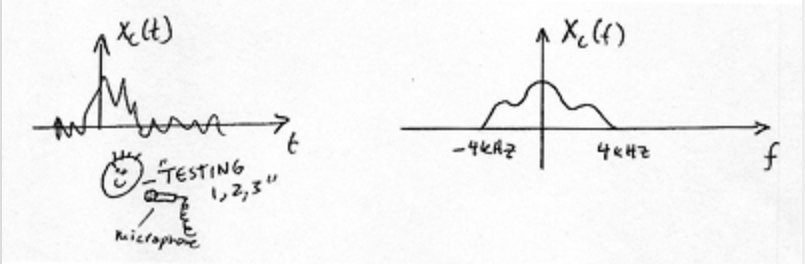
$$\begin{aligned} X(\omega) &= \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c(\Omega - k\Omega_s) \\ &= \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c\left(\frac{\omega - 2\pi k}{T}\right) \end{aligned}$$

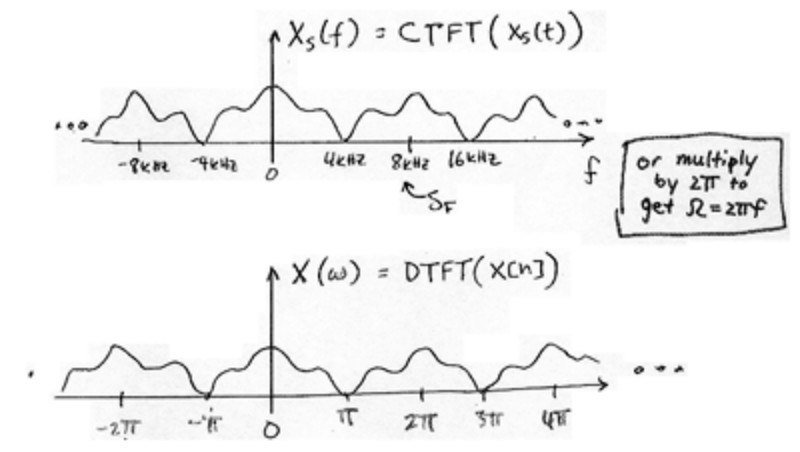
where this last part is 2π-periodic.

Sampling



Example:
Speech
Speech is intelligible if bandlimited by a CT lowpass filter to the band ±4 kHz. We can sample speech as slowly as _____?



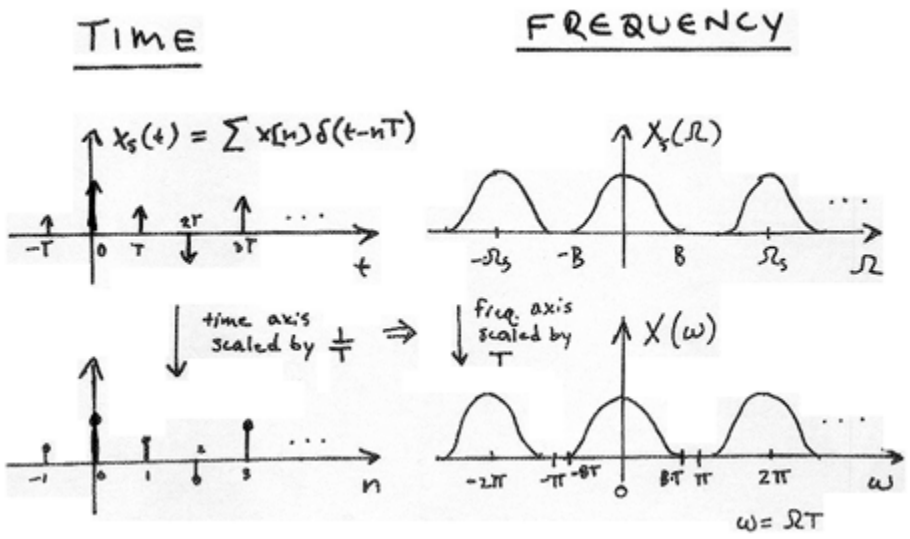


Note that there is no mention of T or Ω_s!

Relating x[n] to sampled x(t)

Recall the following equality:

$$x_s(t) = \sum_{nn} x(nT)\delta(t - nT)$$



Recall the CTFT relation:
Equation:

$$x(\alpha t) \leftrightarrow \frac{1}{\alpha} X \quad \frac{\Omega}{\alpha}$$

where α is a scaling of time and $\frac{1}{\alpha}$ is a scaling in frequency.
Equation:

$$X_s(\Omega) \equiv X(\Omega T)$$

The DFT: Frequency Domain with a Computer Analysis

Introduction

We just covered ideal (and non-ideal) (time) sampling of CT signals. This enabled DT signal processing solutions for CT applications ([link]):

Much of the theoretical analysis of such systems relied on frequency domain representations. How do we carry out these frequency domain analysis on the computer? Recall the following relationships:

$$x[n] \overset{\text{DTFT}}{\longleftrightarrow} X(\omega)$$

$$x(t) \overset{\text{CTFT}}{\longleftrightarrow} X(\Omega)$$

where ω and Ω are continuous frequency variables.

Sampling DTFT

Consider the DTFT of a discrete-time (DT) signal $x[n]$. Assume $x[n]$ is of finite duration N (i.e., an N-point signal).
Equation:

$$X(\omega) = \sum_{n=0}^{N-1} x[n] e^{(-i)\omega n}$$

where $X(\omega)$ is the continuous function that is indexed by the real-valued parameter $-\pi \leq \omega \leq \pi$. The other function, $x[n]$, is a discrete function that

is indexed by integers.

We want to work with $X(\omega)$ on a computer. Why not just **sample** $X(\omega)$?
Equation:

$$\begin{aligned} X[k] &= X\left(\tfrac{2\pi}{N}k\right) \\ &= \textstyle\sum_{n=0}^{N-1} x[n]e^{(-i)2\pi\frac{k}{N}n} \end{aligned}$$

In [link] we sampled at $\omega = \frac{2\pi}{N}k$ where $k = \{0, 1, \ldots, N-1\}$ and $X[k]$ for $k = \{0, \ldots, N-1\}$ is called the **Discrete Fourier Transform (DFT)** of $x[n]$.

Example:
Finite Duration DT Signal
[missing_resource: sec8_fig2.png]
The DTFT of the image in [link] is written as follows:
Equation:

$$X(\omega) = \sum_{n=0}^{N-1} x[n]e^{(-i)\omega n}$$

where ω is any 2π-interval, for example $-\pi \leq \omega \leq \pi$.
Sample X(ω)
[missing_resource: sec8_fig3.png]
where again we sampled at $\omega = \frac{2\pi}{N}k$ where $k = \{0, 1, \ldots, M-1\}$. For example, we take

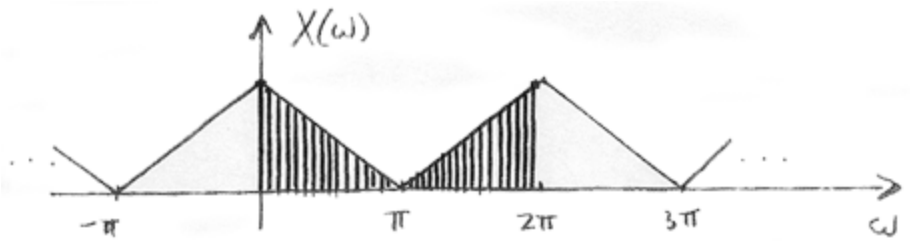
$$M = 10$$

. In the following section we will discuss in more detail how we should choose M, the number of samples in the 2π interval.
(This is precisely how we would plot $X(\omega)$ in Matlab.)

Choosing M

Case 1

Given N (length of $x[n]$), choose $M \gg N$ to obtain a dense sampling of the DTFT ([link]):



Case 2

Choose M as small as possible (to minimize the amount of computation).

In general, we require $M \geq N$ in order to represent all information in

$$\forall n, n = \{0, \ldots, N-1\} : (x[n])$$

Let's concentrate on $M = N$:

$$x[n] \overset{\text{DFT}}{\longleftrightarrow} X[k]$$

for $n = \{0, \ldots, N-1\}$ and $k = \{0, \ldots, N-1\}$

$$\text{numbers} \leftrightarrow \text{N numbers}$$

Discrete Fourier Transform (DFT)

Define
Equation:

$$X[k] \equiv X\left(\frac{2\pi k}{N}\right)$$

where $N = \text{length}\left(x[n]\right)$ and $k = \{0, \ldots, N-1\}$. In this case, $M = N$.
Equation:

DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{(-i)2\pi \frac{k}{N} n}$$

Equation:

Inverse DFT (IDFT)

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{i2\pi \frac{k}{N} n}$$

Interpretation

Represent $x[n]$ in terms of a sum of N complex sinusoids of amplitudes $X[k]$ and frequencies

$$\forall k, k \in \{0, \ldots, N-1\} : \left(\omega_k = \frac{2\pi k}{N}\right)$$

Note: Fourier Series with fundamental frequency $\frac{2\pi}{N}$

Remark 1

IDFT treats $x[n]$ as though it were N-periodic.

Equation:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{i 2\pi \frac{k}{N} n}$$

where $n \in \{0, \ldots, N-1\}$

Exercise:

Problem: What about other values of n?

Solution:

$$x[n+N] = ???$$

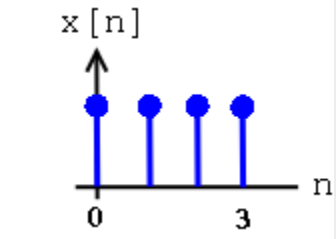
Remark 2

Proof that the IDFT inverts the DFT for $n \in \{0, \ldots, N-1\}$

Equation:

$$\begin{aligned} \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{i 2\pi \frac{k}{N} n} &= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} x[m] e^{(-i) 2\pi \frac{k}{N} m} e^{i 2\pi \frac{k}{N} n} \\ &= ??? \end{aligned}$$

Example:
Computing DFT
Given the following discrete-time signal ([link]) with $N = 4$, we will compute the DFT using two different methods (the DFT Formula and Sample DTFT):



1. DFT Formula
   Equation:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] e^{(-i)2\pi \frac{k}{N} n} \\ &= 1 + e^{(-i)2\pi \frac{k}{4}} + e^{(-i)2\pi \frac{k}{4} 2} + e^{(-i)2\pi \frac{k}{4} 3} \\ &= 1 + e^{(-i)\frac{\pi}{2} k} + e^{(-i)\pi k} + e^{(-i)\frac{3}{2}\pi k} \end{aligned}$$

   Using the above equation, we can solve and get the following results:

$$x[0] = 4$$

$$x[1] = 0$$

$$x[2] = 0$$

$$x[3] = 0$$

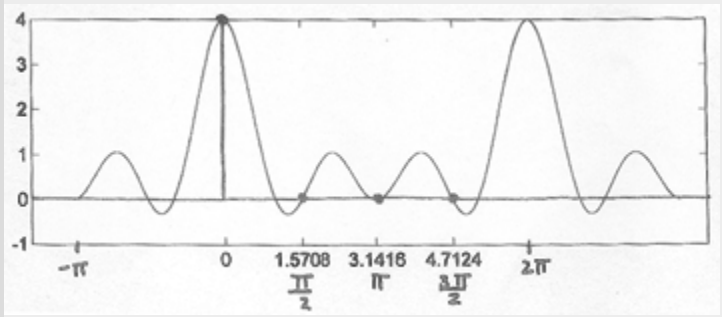
2. Sample DTFT. Using the same figure, [link], we will take the DTFT of the signal and get the following equations:
   Equation:

$$\begin{aligned} X(\omega) &= \sum_{n=0}^{3} e^{(-i)\omega n} \\ &= \frac{1 - e^{(-i)4\omega}}{1 - e^{(-i)\omega}} \\ &= ??? \end{aligned}$$

   Our sample points will be:

$$\omega_k = \frac{2\pi k}{4} = \frac{\pi}{2} k$$

where $k = \{0, 1, 2, 3\}$ ([link]).

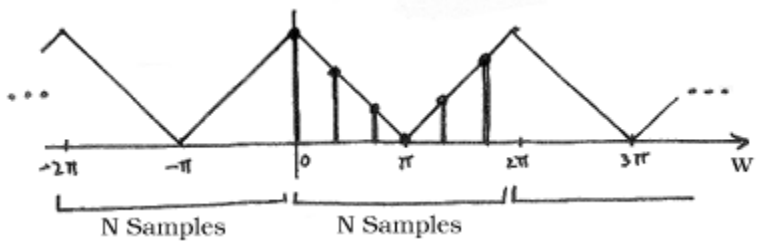


Periodicity of the DFT

DFT $X[k]$ consists of **samples** of DTFT, so $X(\omega)$, a 2π-periodic DTFT signal, can be converted to $X[k]$, an N-periodic DFT.
Equation:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{(-i)2\pi \frac{k}{N} n}$$

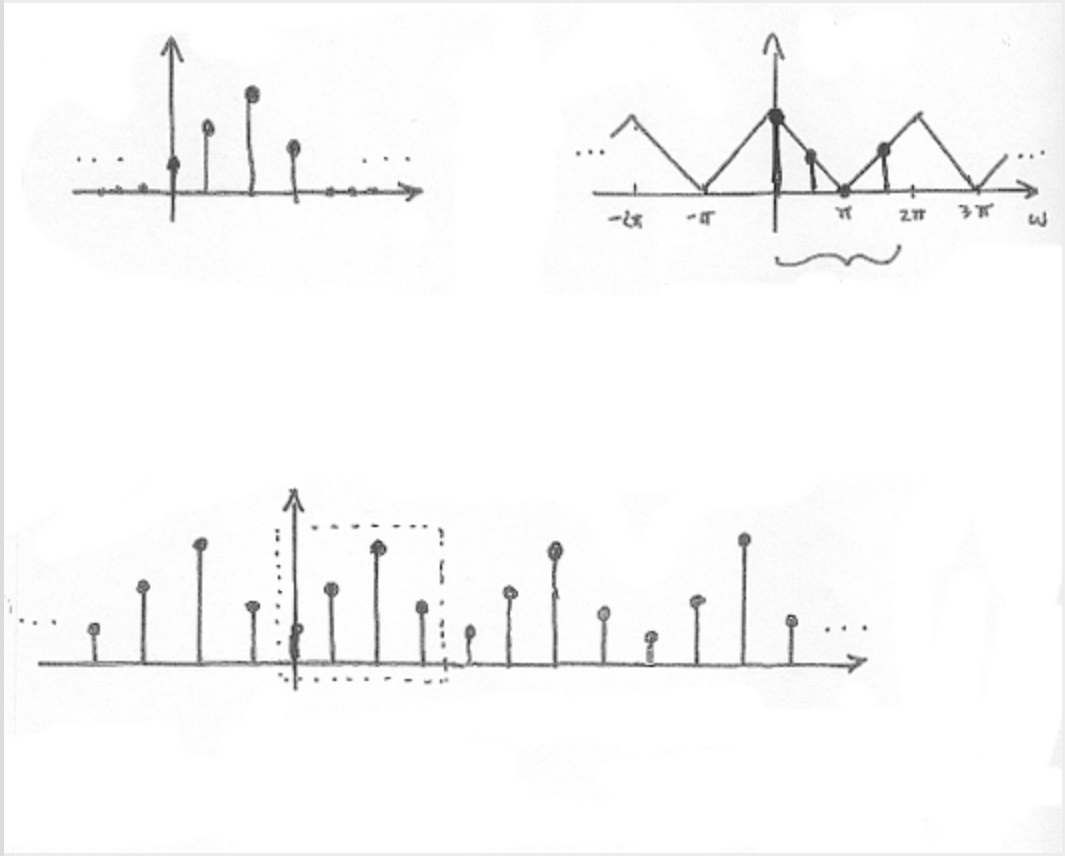
where $e^{(-i)2\pi \frac{k}{N} n}$ is an N-periodic basis function (See [link]).



Also, recall,
Equation:

$$
\begin{aligned}
x[n] &= \tfrac{1}{N}\sum_{n=0}^{N-1} X[k] e^{i2\pi\frac{k}{N}n} \\
&= \tfrac{1}{N}\sum_{n=0}^{N-1} X[k] e^{i2\pi\frac{k}{N}(n+mN)} \\
&= ???
\end{aligned}
$$

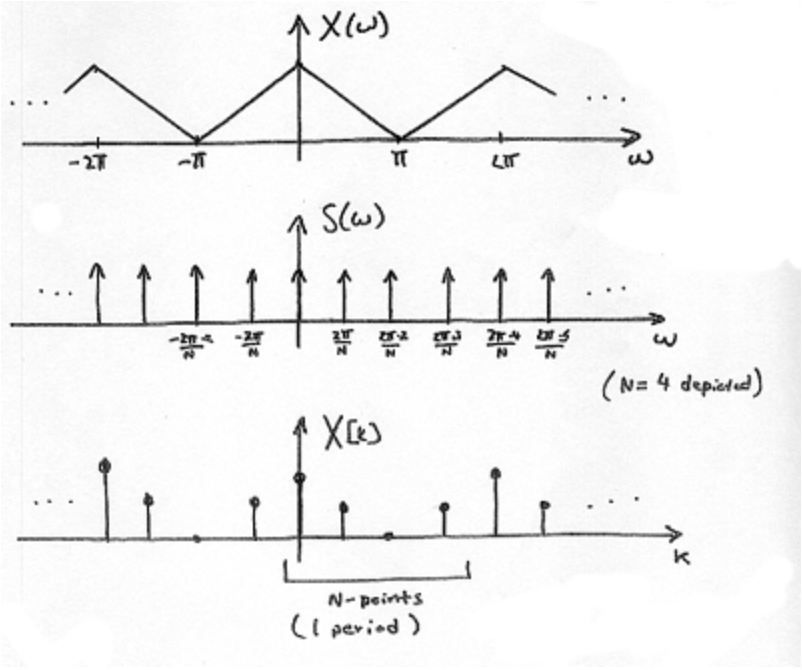
Example:
Illustration



Note: When we deal with the DFT, we need to remember that, in effect, this treats the signal as an N-periodic sequence.

A Sampling Perspective

Think of sampling the continuous function $X(\omega)$, as depicted in [link]. $S(\omega)$ will represent the sampling function applied to $X(\omega)$ and is illustrated in [link] as well. This will result in our discrete-time sequence, $X[k]$.



Note: Remember the multiplication in the frequency domain is equal to convolution in the time domain!

Inverse DTFT of S(ω)

Equation:

$$\sum_{k=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi k}{N}\right)$$

Given the above equation, we can take the DTFT and get the following equation:
Equation:

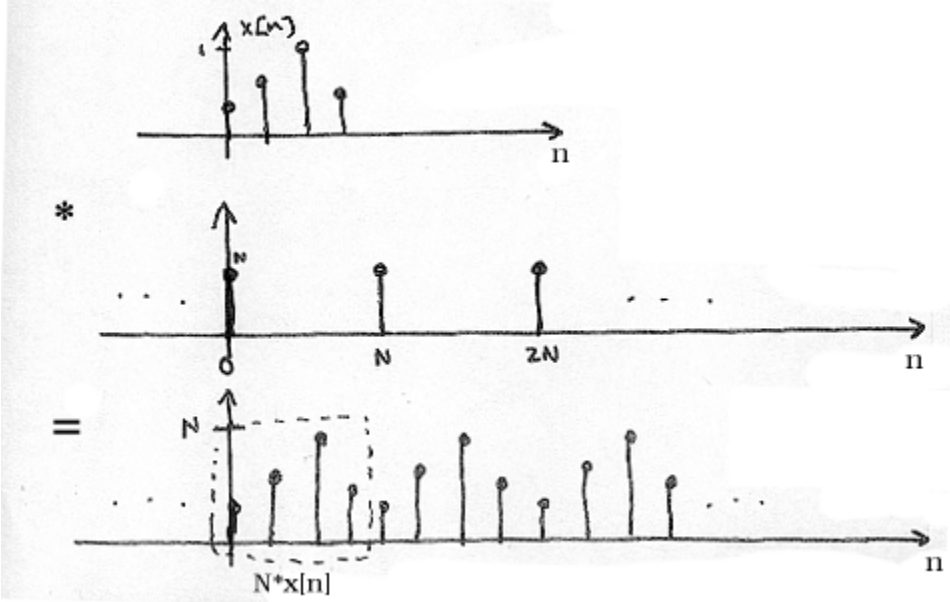
$$N \sum_{m=-\infty}^{\infty} \delta[n - mN] \equiv S[n]$$

Exercise:

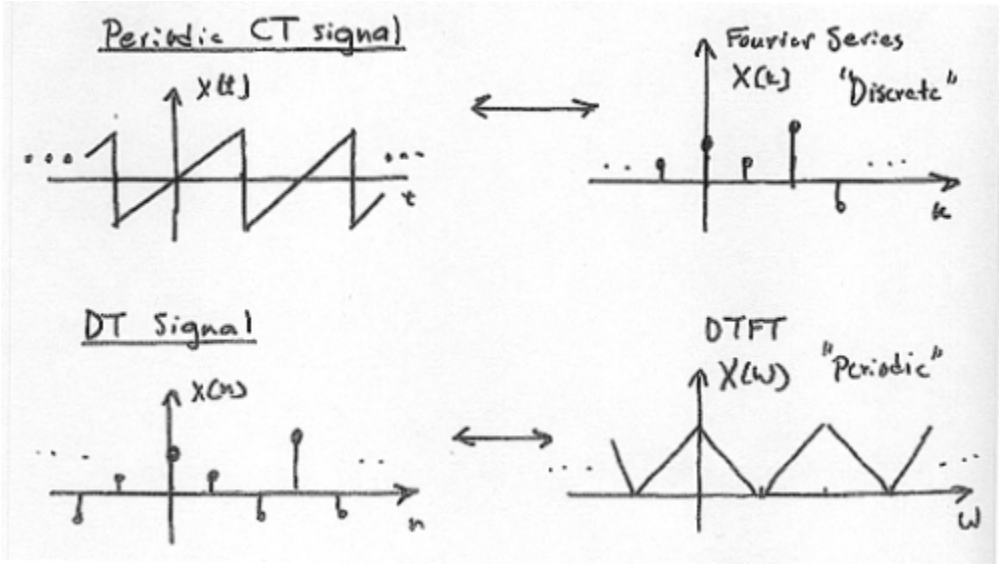
> **Problem:** Why does [link] equal $S[n]$?

> **Solution:**
>
> $S[n]$ is N-periodic, so it has the following Fourier Series:
> **Equation:**
>
> $$\begin{aligned} c_k &= \frac{1}{N} \int_{-\frac{N}{2}}^{\frac{N}{2}} \delta[n] e^{(-i)2\pi \frac{k}{N} n} \, \mathrm{d}\, n \\ &= \frac{1}{N} \end{aligned}$$
>
> **Equation:**
>
> $$S[n] = \sum_{k=-\infty}^{\infty} e^{(-i)2\pi \frac{k}{N} n}$$
>
> where the DTFT of the exponential in the above equation is equal to $\delta\left(\omega - \frac{2\pi k}{N}\right)$.

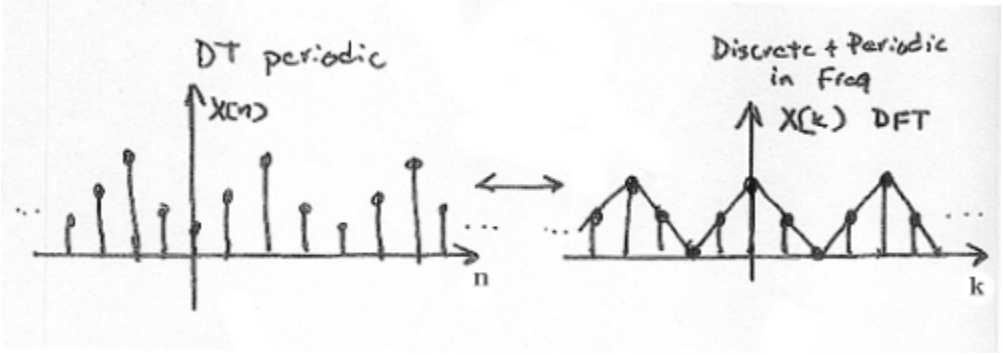
So, in the time-domain we have ([link]):



Connections



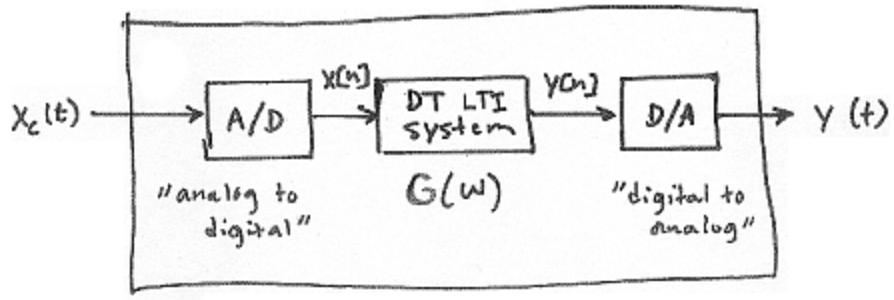
Combine signals in [link] to get signals in [link].


DT periodic
X[n]
n
Discrete & Periodic
in Freq
X(k) DFT
k

Discrete-Time Processing of CT Signals

DT Processing of CT Signals

DSP System



Analysis

Equation:

$$Y_c(\Omega) = H_{\text{LP}}(\Omega)Y(\Omega T)$$

where we know that $Y(\omega) = X(\omega)G(\omega)$ and $G(\omega)$ is the frequency response of the DT LTI system. Also, remember that

$$\omega \equiv \Omega T$$

So,
Equation:

$$Y_c(\Omega) = H_{\text{LP}}(\Omega)G(\Omega T)X(\Omega T)$$

where $Y_c(\Omega)$ and $H_{\text{LP}}(\Omega)$ are CTFTs and $G(\Omega T)$ and $X(\Omega T)$ are DTFTs.

Note:

$$X(\omega) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} X_c\left(\frac{\omega - 2\pi k}{T}\right)$$

OR

$$X(\Omega T) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} X_c(\Omega - k\Omega_s)$$

Therefore our final output signal, $Y_c(\Omega)$, will be:
Equation:

$$Y_c(\Omega) = H_{\text{LP}}(\Omega) G(\Omega T) \left(\frac{2\pi}{T} \sum_{k=-\infty}^{\infty} X_c(\Omega - k\Omega_s) \right)$$

Now, if $X_c(\Omega)$ is bandlimited to $\left[-\frac{\Omega_s}{2}, \frac{\Omega_s}{2}\right]$ and we use the usual lowpass reconstruction filter in the D/A, [link]:

[missing_resource: sec9_fig2.png]

Then,
Equation:

$$Y_c(\Omega) = \begin{cases} G(\Omega T) X_c(\Omega) & \text{if} \quad |\Omega| < \frac{\Omega_s}{2} \\ 0 \quad \text{otherwise} \end{cases}$$

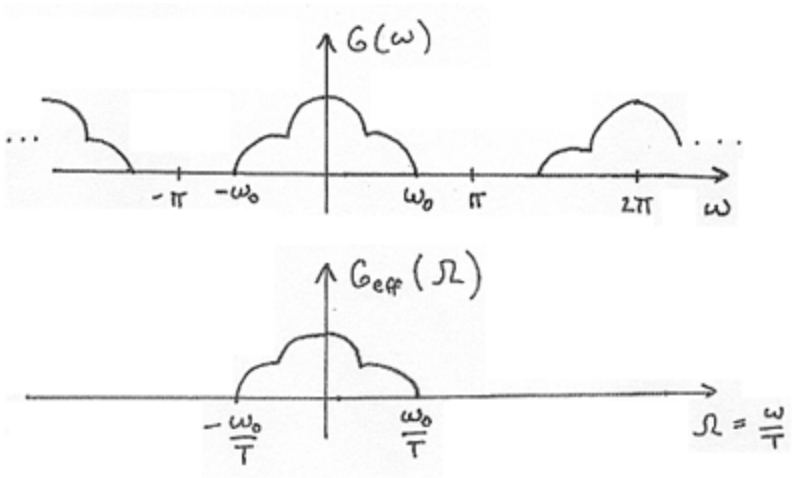
Summary

For bandlimited signals sampled at or above the Nyquist rate, we can relate the input and output of the DSP system by:
Equation:

$$Y_c(\Omega) = G_{\text{eff}}(\Omega) X_c(\Omega)$$

where

$$G_{\text{eff}}(\Omega) = \begin{cases} G(\Omega T) \ \text{ if } \ |\Omega| < \frac{\Omega_s}{2} \\ 0 \ \text{ otherwise} \end{cases}$$



Note

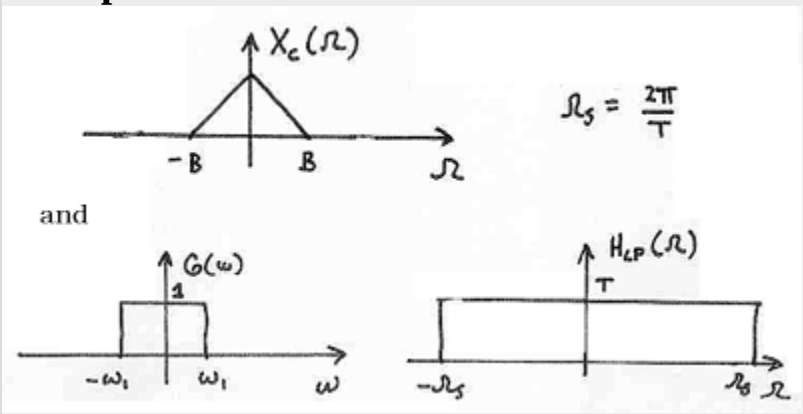
$G_{\text{eff}}(\Omega)$ is LTI if and only if the following two conditions are satisfied:

1. $G(\omega)$ is LTI (in DT).
2. $X_c(T)$ is bandlimited and sampling rate equal to or greater than Nyquist. For example, if we had a simple pulse described by

$$X_c(t) = u(t - T_0) - u(t - T_1)$$

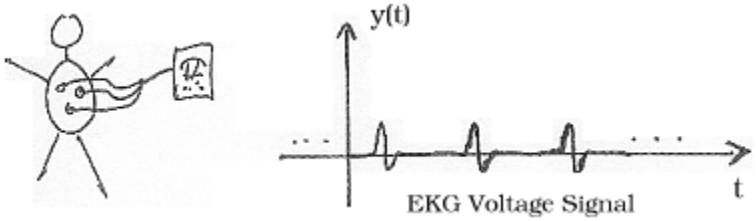
where $T_1 > T_0$. If the sampling period $T > T_1 - T_0$, then some samples might "miss" the pulse while others might not be "missed." This is what we term **time-varying behavior**.

Example:

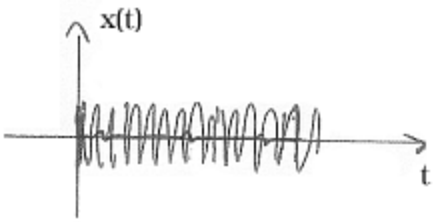


If $\frac{2\pi}{T} > 2B$ and $\omega_1 < BT$, determine and sketch $Y_c(\Omega)$ using [link].

Application: 60Hz Noise Removal

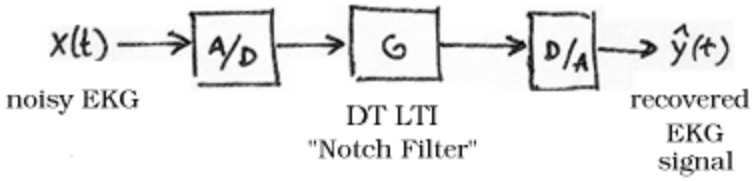


Unfortunately, in real-world situations electrodes also pick up ambient 60 Hz signals from lights, computers, etc.. In fact, usually this "60 Hz noise" is much greater in amplitude than the EKG signal shown in [link]. [link] shows the EKG signal; it is barely noticeable as it has become overwhelmed by noise.



Our EKG signal, $y(t)$, is overwhelmed by noise.

DSP Solution



[missing_resource: sec9_fig7b.png]

Sampling Period/Rate

First we must note that $|Y(\Omega)|$ is **bandlimited** to ±60 Hz. Therefore, the minimum rate should be 120 Hz. In order to get the best results we should set

$$f_s = 240\,\text{Hz}$$

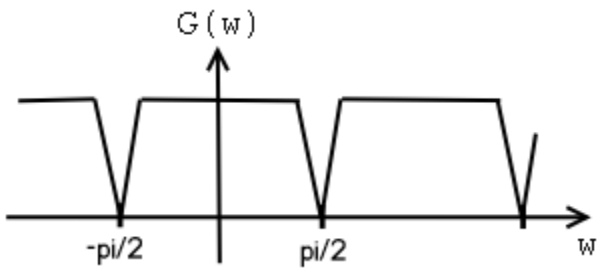
.

$$\Omega_s = 2\pi \times \left(240 \frac{\text{rad}}{s} \right)$$

[missing_resource: sec9_fig8.png]

Digital Filter

Therefore, we want to design a digital filter that will remove the 60Hz component and preserve the rest.



Short Time Fourier Transform

Short Time Fourier Transform

The Fourier transforms (FT, DTFT, DFT, etc.) do not clearly indicate how the frequency content of a signal changes over time.

That information is hidden in the phase - it is not revealed by the plot of the magnitude of the spectrum.

> **Note:** To see how the frequency content of a signal changes over time, we can cut the signal into blocks and compute the spectrum of each block.

To improve the result,

1. blocks are overlapping
2. each block is multiplied by a window that is tapered at its endpoints.

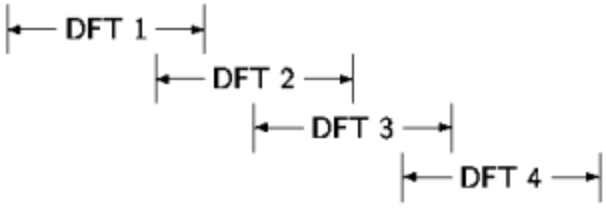
Several parameters must be chosen:

- Block length, R.
- The type of window.
- Amount of overlap between blocks. ([link])
- Amount of zero padding, if any.

STFT: Overlap Parameter

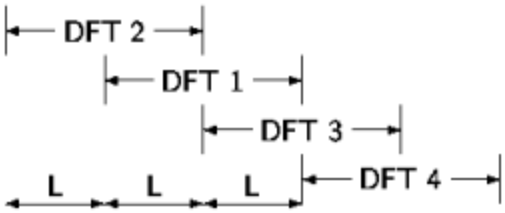
NO OVERLAP

R/4 OVERLAP



R/2 OVERLAP

The parameter L



L is the number of samples between adjacent blocks.

The short-time Fourier transform is defined as
Equation:

$$\begin{aligned} X(\omega, m) &= \operatorname{STFT}(x(n)) \coloneqq \operatorname{DTFT}(x(n-m)w(n)) \\ &= \sum_{n=-\infty}^{\infty} x(n-m)w(n)e^{-(i\omega n)} \\ &= \sum_{n=0}^{R-1} x(n-m)w(n)e^{-(i\omega n)} \end{aligned}$$

where $w(n)$ is the window function of length R.

1. The STFT of a signal $x(n)$ is a function of two variables: time and frequency.
2. The block length is determined by the support of the window function $w(n)$.
3. A graphical display of the magnitude of the STFT, $|X(\omega, m)|$, is called the **spectrogram** of the signal. It is often used in speech processing.
4. The STFT of a signal is invertible.
5. One can choose the block length. A long block length will provide higher frequency resolution (because the main-lobe of the window function will be narrow). A short block length will provide higher time resolution because less averaging across samples is performed for each STFT value.
6. A **narrow-band spectrogram** is one computed using a relatively long block length R, (long window function).
7. A **wide-band spectrogram** is one computed using a relatively short block length R, (short window function).

Sampled STFT

To numerically evaluate the STFT, we sample the frequency axis ω in N equally spaced samples from $\omega = 0$ to $\omega = 2\pi$.
Equation:

$$\forall k, 0 \leq k \leq N-1 : \left(\omega_k = \frac{2\pi}{N} k\right)$$

We then have the discrete STFT,
Equation:

$$\begin{aligned} X^d(k,m) \coloneqq X\left(\tfrac{2\pi}{N}k, m\right) &= \textstyle\sum_{n=0}^{R-1} x(n-m)w(n)e^{-(i\omega n)} \\ &= \textstyle\sum_{n=0}^{R-1} x(n-m)w(n){W_N}^{-(kn)} \\ &= \mathrm{DFT}_N\left(x(n-m)w(n)|_{n=0}^{R-1}, 0, \ldots 0\right) \end{aligned}$$

where $0,\ldots 0$ is $N-R$.

In this definition, the overlap between adjacent blocks is $R-1$. The signal is shifted along the window one sample at a time. That generates more points than is usually needed, so we also sample the STFT along the time direction. That means we usually evaluate

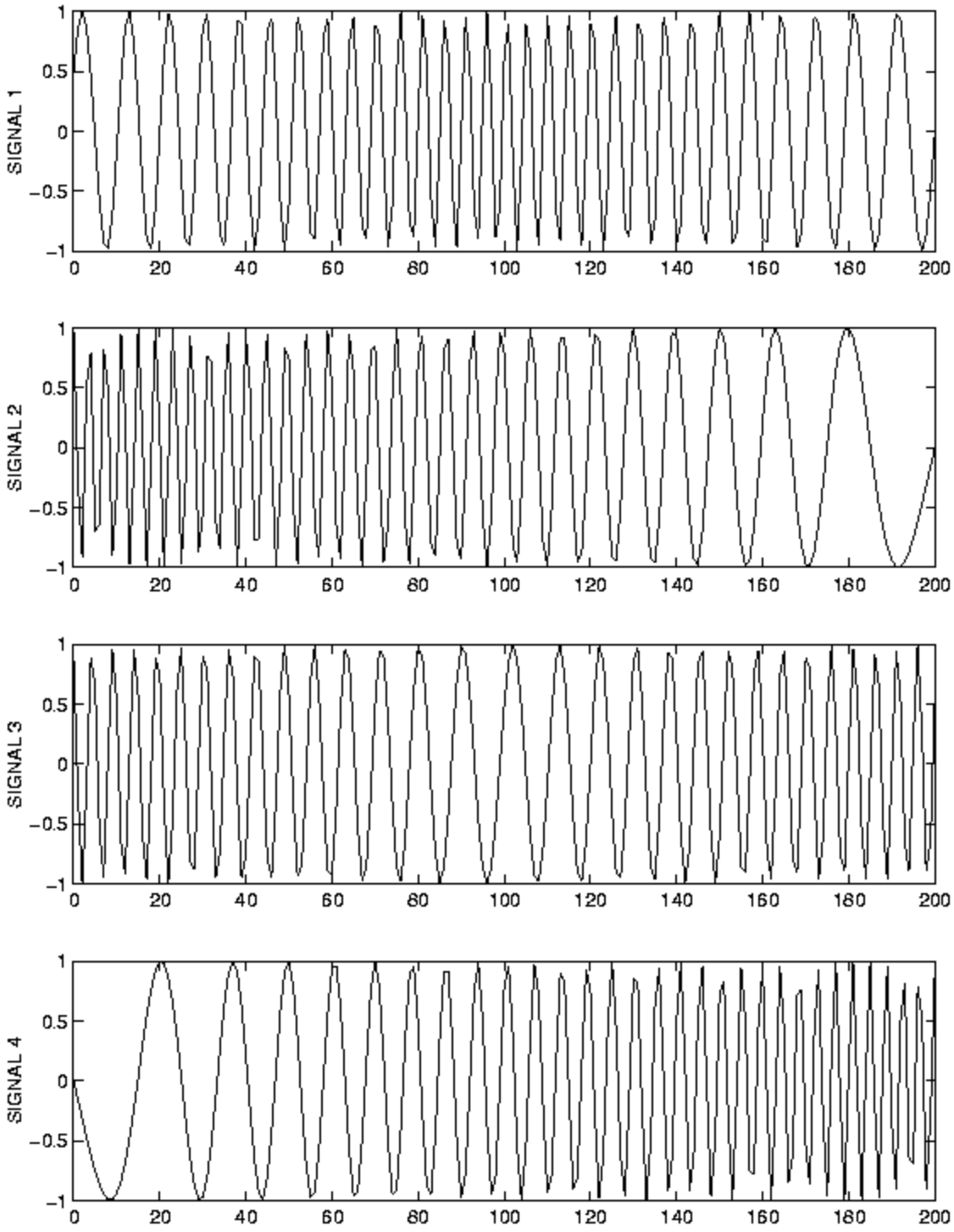
$$X^d(k, Lm)$$

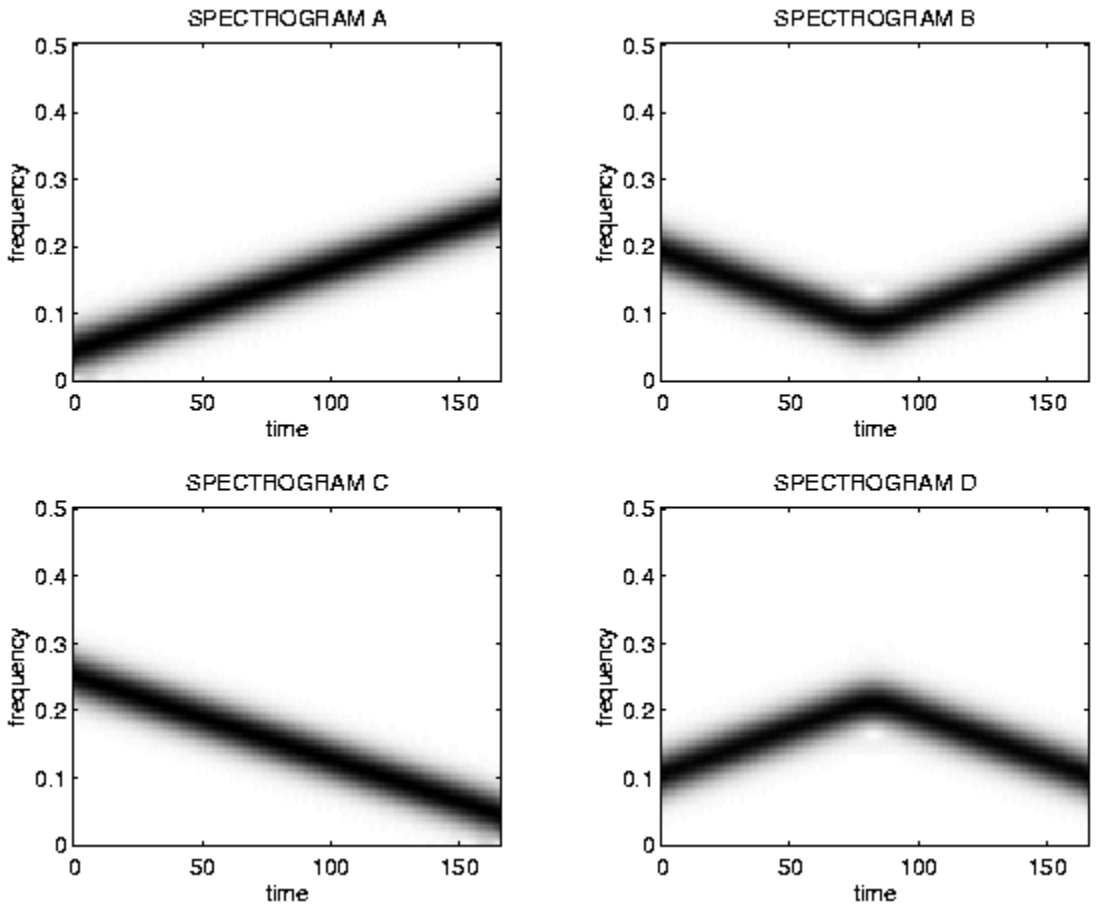
where L is the time-skip. The relation between the time-skip, the number of overlapping samples, and the block length is

$$\text{Overlap} = R - L$$

Exercise:

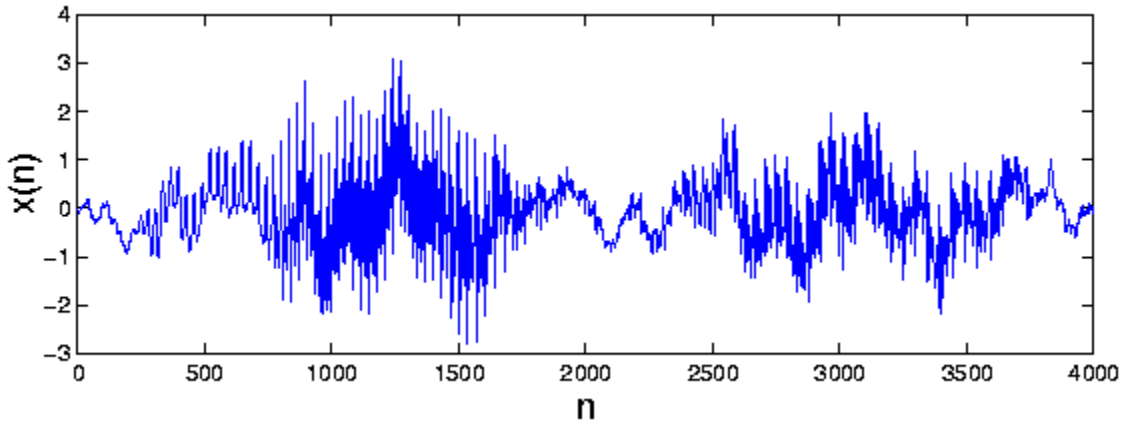
Problem: Match each signal to its spectrogram in [link].

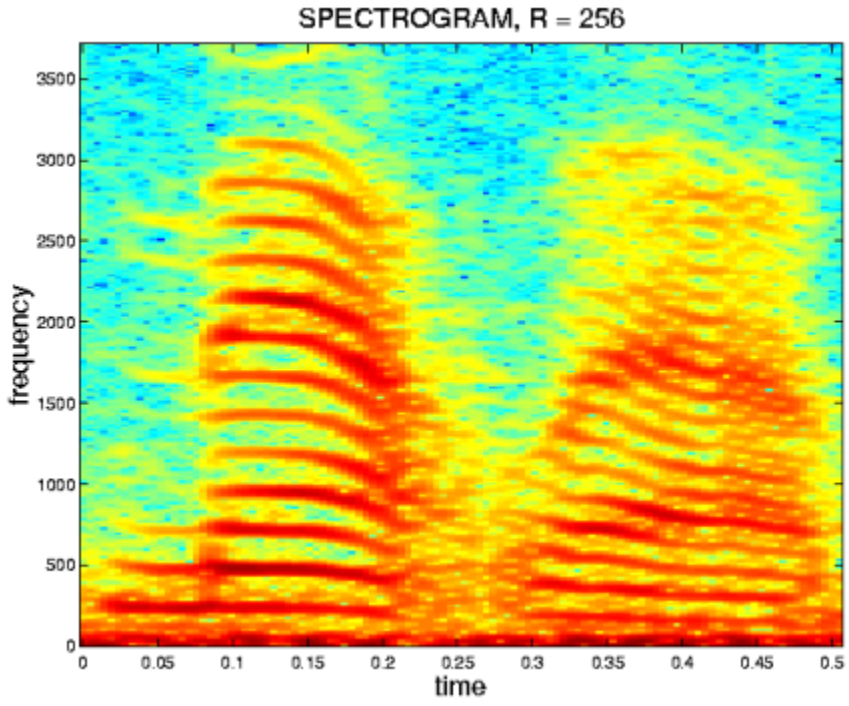

SIGNAL 1
SIGNAL 2
SIGNAL 3
SIGNAL 4
1
0.5
0
−0.5
−1
0
20
40
60
80
100
120
140
160
180
200



Solution:

Spectrogram Example





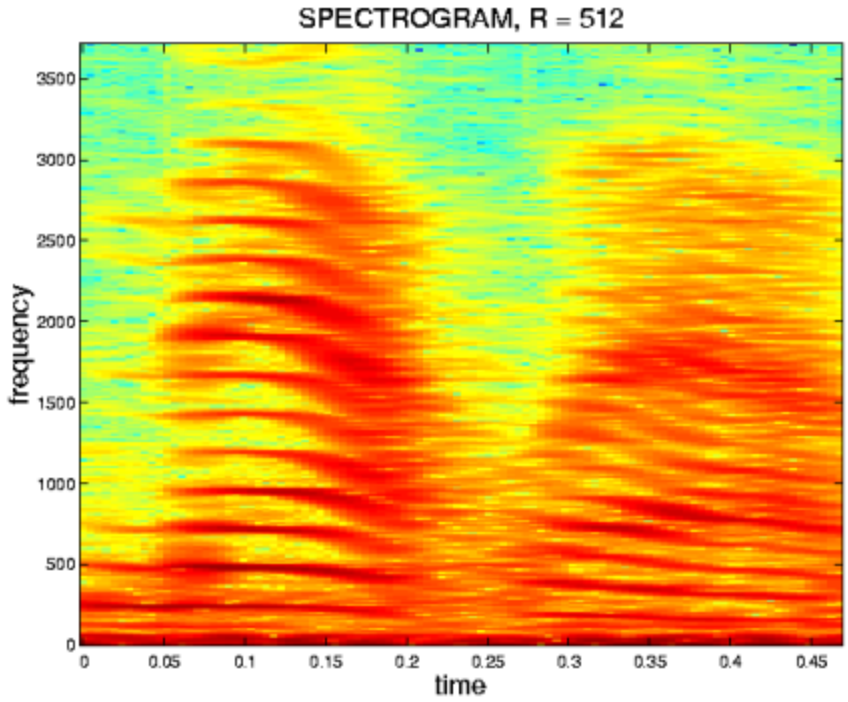
The matlab program for producing the figures above ([link] and [link]).

```
% LOAD DATA
load mtlb;
x = mtlb;
```

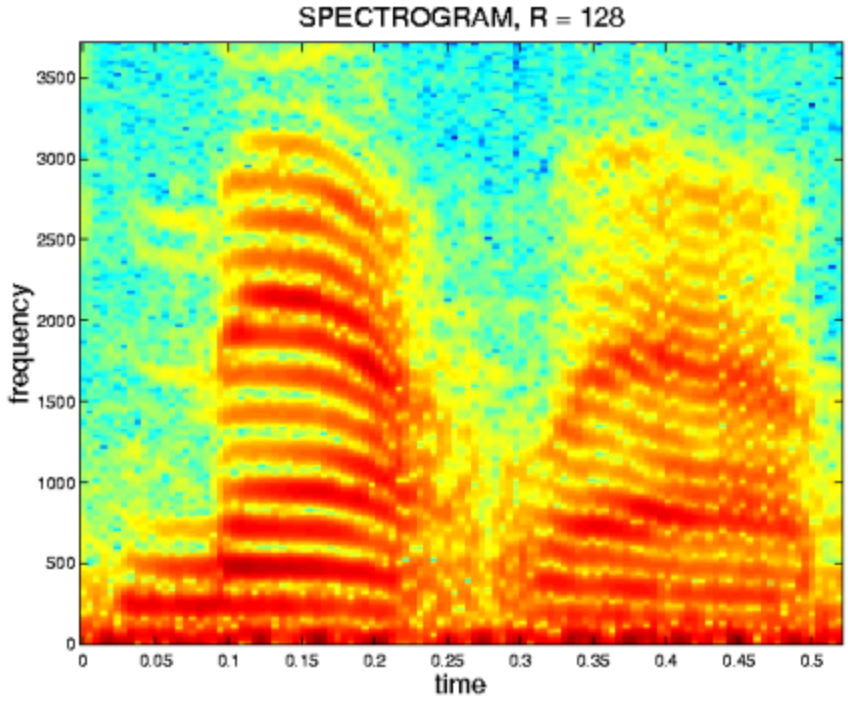
```
figure(1), clf
plot(0:4000,x)
xlabel('n')
ylabel('x(n)')

% SET PARAMETERS
R = 256;                % R: block length
window = hamming(R);    % window function
of length R
N = 512;                % N: frequency
discretization
L = 35;                 % L: time lapse
between blocks
fs = 7418;              % fs: sampling
frequency
overlap = R - L;

% COMPUTE SPECTROGRAM
[B,f,t] =
specgram(x,N,fs,window,overlap);

% MAKE PLOT
figure(2), clf
imagesc(t,f,log10(abs(B)));
colormap('jet')
axis xy
xlabel('time')
ylabel('frequency')
title('SPECTROGRAM, R = 256')
```

Effect of window length R

Narrow-band spectrogram: better frequency resolution

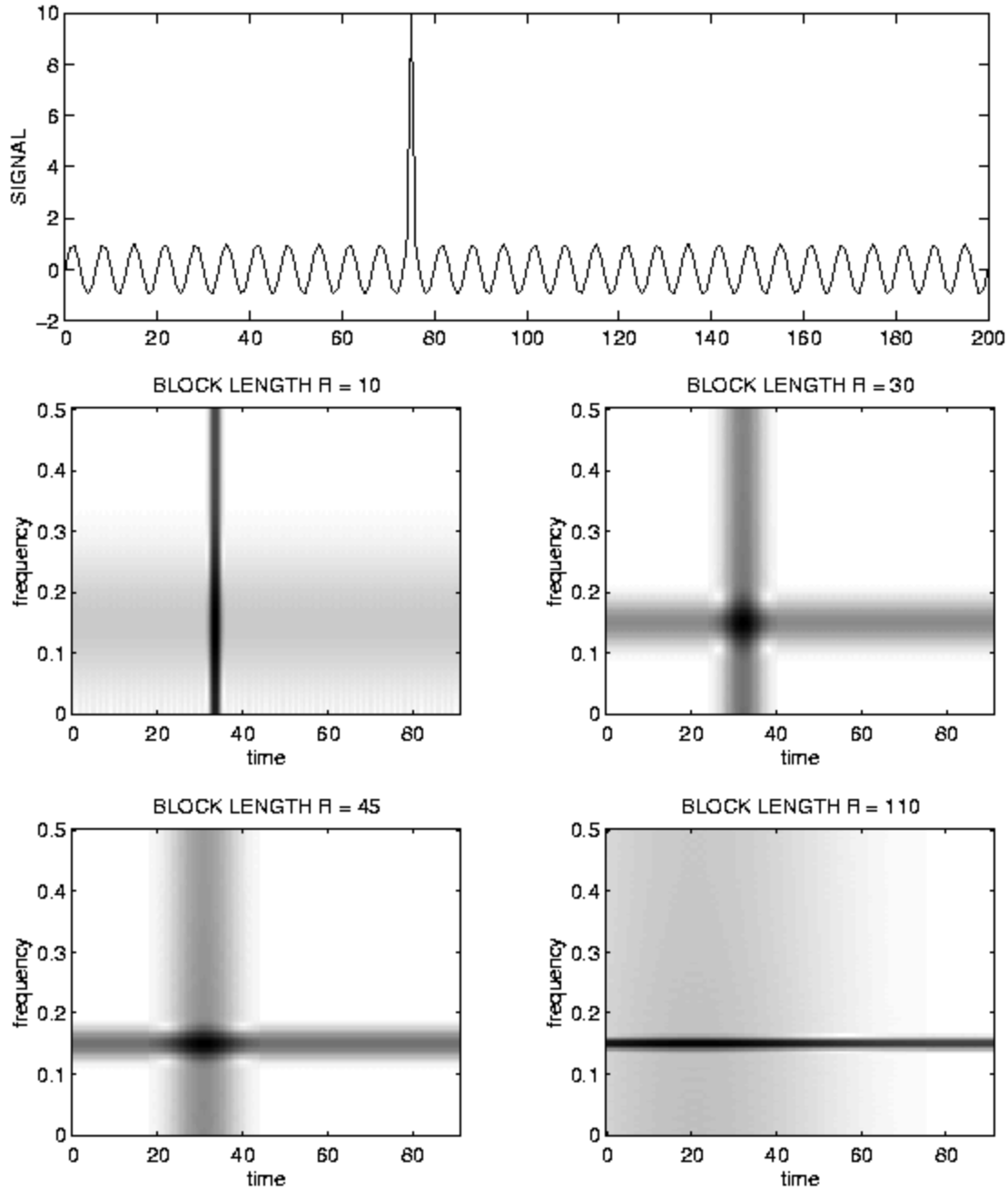


Wide-band spectrogram: better time resolution



Here is another example to illustrate the frequency/time resolution trade-off (See figures - [link], [link], and [link]).

Effect of Window Length R



Effect of L and N

A spectrogram is computed with different parameters:

$$L \in \{1, 10\}$$

$$N \in \{32, 256\}$$

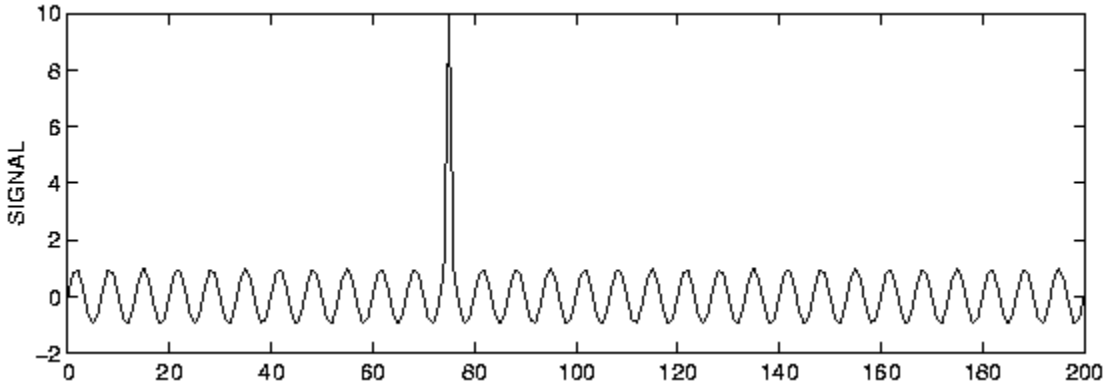
- L = time lapse between blocks.
- N = FFT length (Each block is zero-padded to length N.)

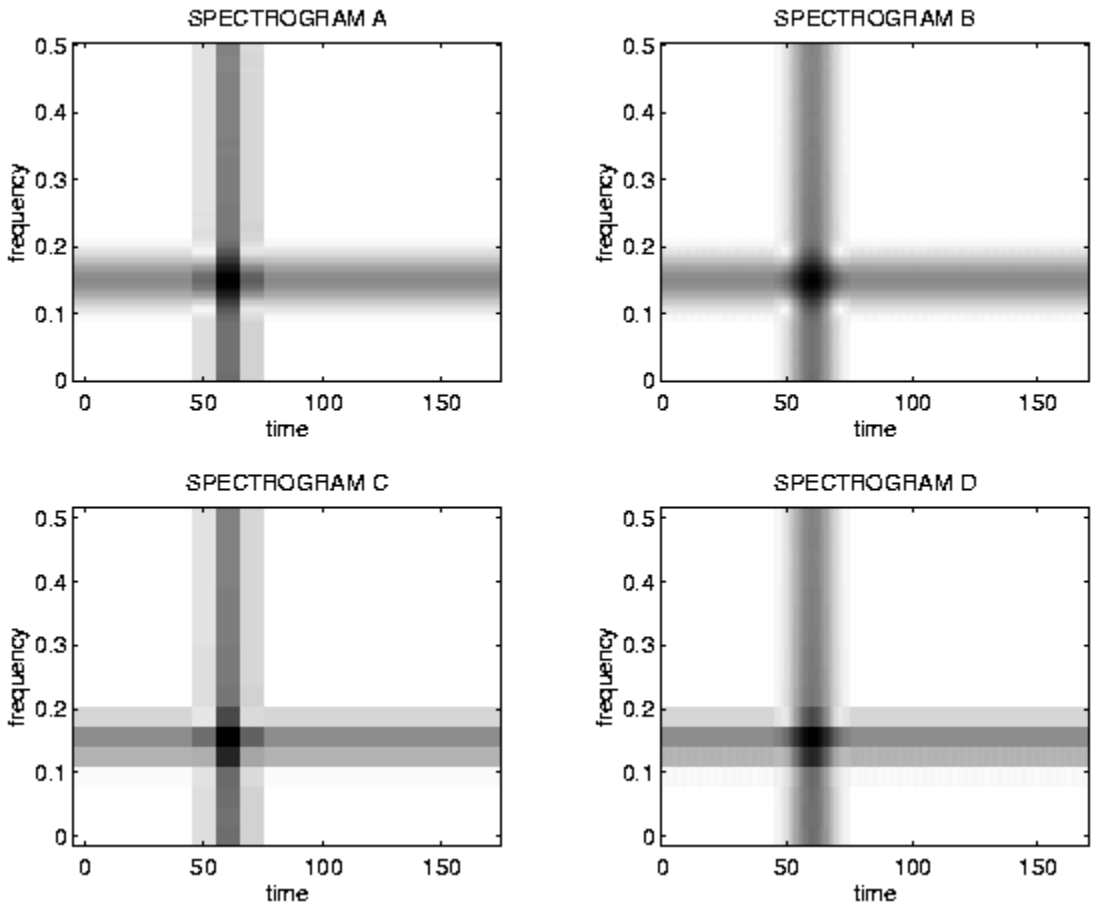
In each case, the block length is 30 samples.

Exercise:

Problem:

For each of the four spectrograms in [link] can you tell what L and N are?





Solution:

L and N do not effect the time resolution or the frequency resolution. They only affect the 'pixelation'.

Effect of R and L

Shown below are four spectrograms of the same signal. Each spectrogram is computed using a different set of parameters.

$$R \in \{120, 256, 1024\}$$

$$L \in \{35, 250\}$$

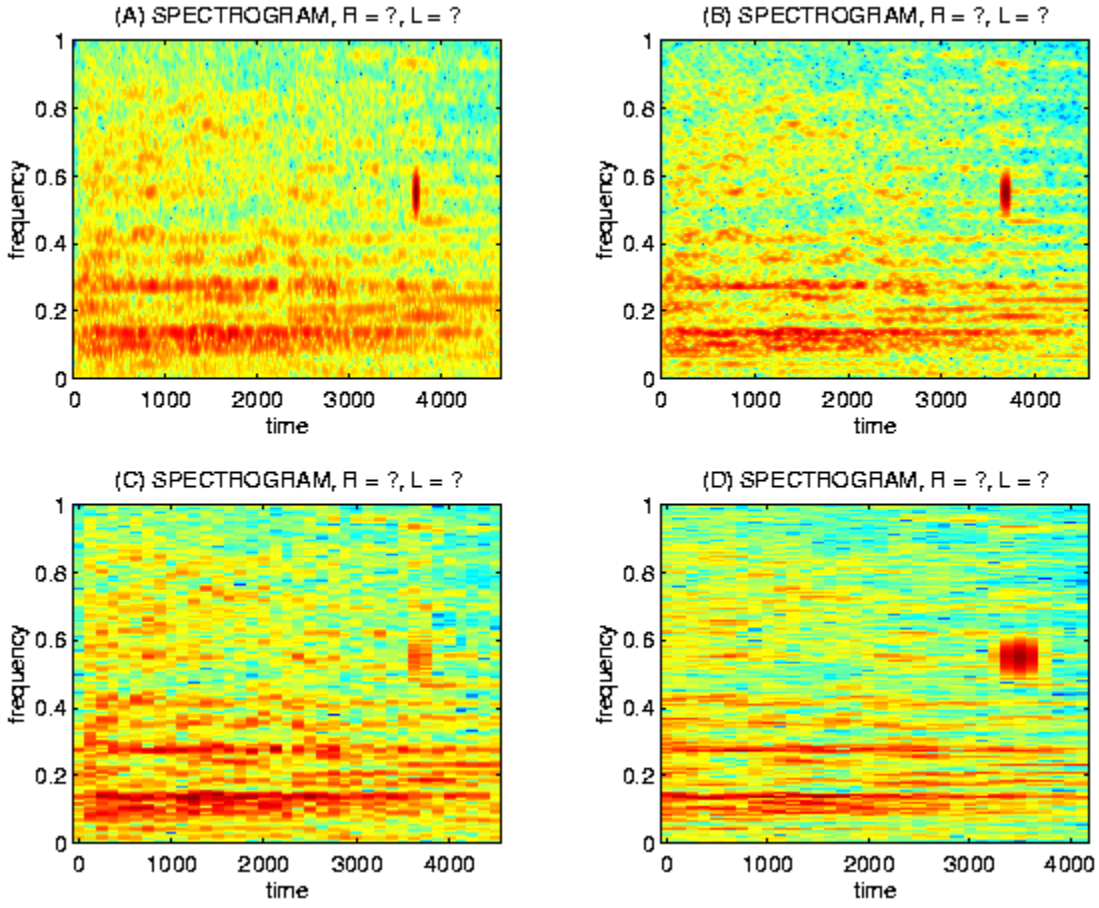
where

- R = block length
- L = time lapse between blocks.

Exercise:

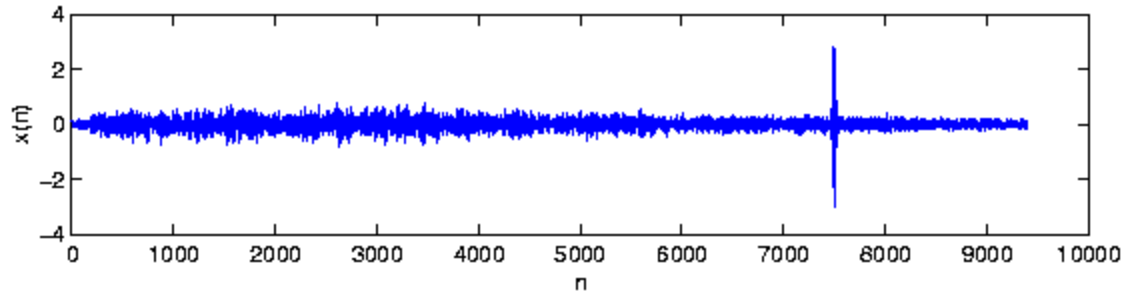
Problem:

For each of the four spectrograms in [link], match the above values of L and R.



Solution:

If you like, you may listen to this signal with the `soundsc` command; the data is in the file: `stft_data.m`. Here is a figure of the signal.


4
2
0
−2
−4
x(n)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
n

Spectrograms

Spectrograms visually represent the speach signal, and the calculation of the Spectrogram is briefly explained.

We know how to acquire analog signals for digital processing (pre-filtering, sampling, and A/D conversion) and to compute spectra of discrete-time signals (using the FFT algorithm), let's put these various components together to learn how the spectrogram shown in [link], which is used to analyze speech, is calculated. The speech was sampled at a rate of 11.025 kHz and passed through a 16-bit A/D converter.

Note: Music compact discs (CDs) encode their signals at a sampling rate of 44.1 kHz. We'll learn the rationale for this number later. The 11.025 kHz sampling rate for the speech is 1/4 of the CD sampling rate, and was the lowest available sampling rate commensurate with speech signal bandwidths available on my computer.

Exercise:

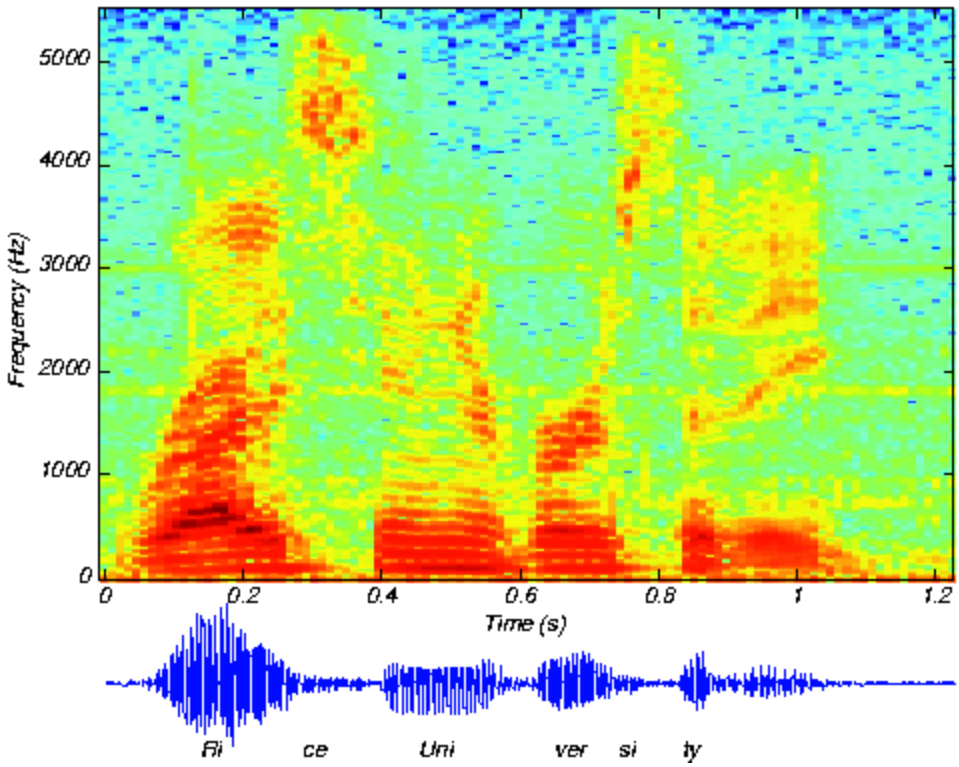
Problem:

Looking at [link] the signal lasted a little over 1.2 seconds. How long was the sampled signal (in terms of samples)? What was the datarate during the sampling process in bps (bits per second)? Assuming the computer storage is organized in terms of bytes (8-bit quantities), how many bytes of computer memory does the speech consume?

Solution:

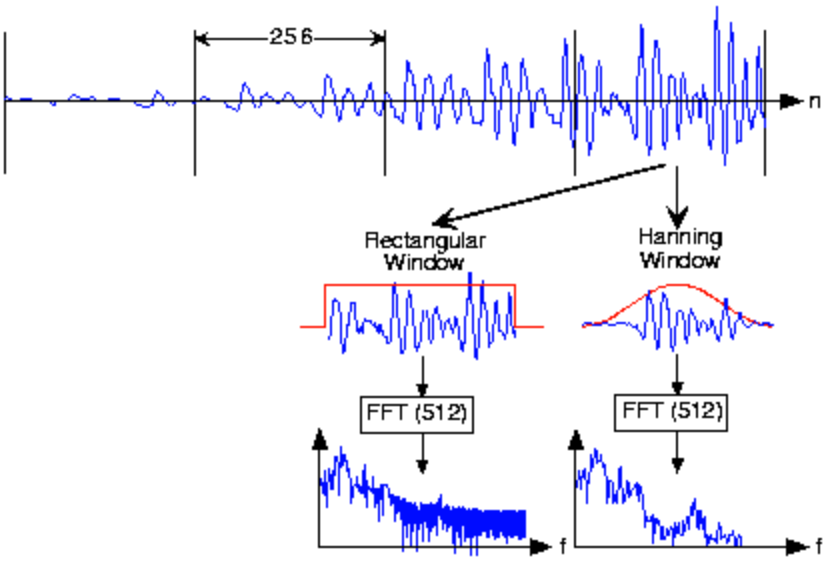
Number of samples equals $1.2 \times 11025 = 13230$. The datarate is $11025 \times 16 = 176.4$ kbps. The storage required would be 26460 bytes.

Speech Spectrogram



The resulting discrete-time signal, shown in the bottom of [link], clearly changes its character with time. To display these spectral changes, the long signal was sectioned into **frames**: comparatively short, contiguous groups of samples. Conceptually, a Fourier transform of each frame is calculated using the FFT. Each frame is not so long that significant signal variations are retained within a frame, but not so short that we lose the signal's spectral character. Roughly speaking, the speech signal's spectrum is evaluated over successive time segments and stacked side by side so that the x-axis corresponds to time and the y-axis frequency, with color indicating the spectral amplitude.

An important detail emerges when we examine each framed signal ([link]).
Spectrogram Hanning vs. Rectangular



The top waveform is a segment 1024 samples long taken from the beginning of the "Rice University" phrase. Computing [link] involved creating frames, here demarked by the vertical lines, that were 256 samples long and finding the spectrum of each. If a rectangular window is applied (corresponding to extracting a frame from the signal), oscillations appear in the spectrum (middle of bottom row). Applying a Hanning window gracefully tapers the signal toward frame edges, thereby yielding a more accurate computation of the signal's spectrum at that moment of time.

At the frame's edges, the signal may change very abruptly, a feature not present in the original signal. A transform of such a segment reveals a curious oscillation in the spectrum, an artifact directly related to this sharp amplitude change. A better way to frame signals for spectrograms is to apply a **window**: Shape the signal values within a frame so that the signal decays gracefully as it nears the edges. This shaping is accomplished by multiplying the framed signal by the sequence $w(n)$. In sectioning the signal, we essentially applied a rectangular window: $w(n) = 1$,

$0 \leq n \leq N-1$. A much more graceful window is the **Hanning window**; it has the cosine shape $w(n) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N}\right)\right)$. As shown in [link], this shaping greatly reduces spurious oscillations in each frame's spectrum. Considering the spectrum of the Hanning windowed frame, we find that the oscillations resulting from applying the rectangular window obscured a formant (the one located at a little more than half the Nyquist frequency).

Exercise:

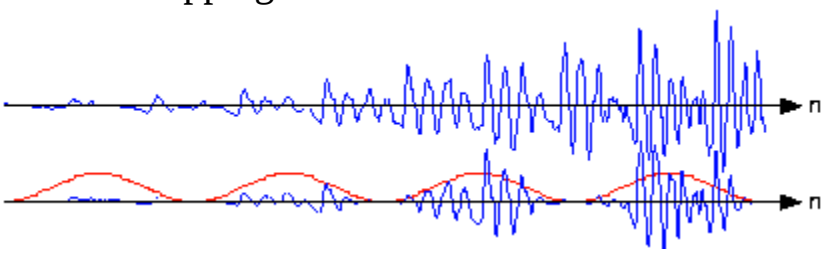
Problem:

What might be the source of these oscillations? To gain some insight, what is the length-$2N$ discrete Fourier transform of a length-N pulse? The pulse emulates the rectangular window, and certainly has edges. Compare your answer with the length-$2N$ transform of a length-N Hanning window.

Solution:

The oscillations are due to the boxcar window's Fourier transform, which equals the sinc function.

Non-overlapping windows

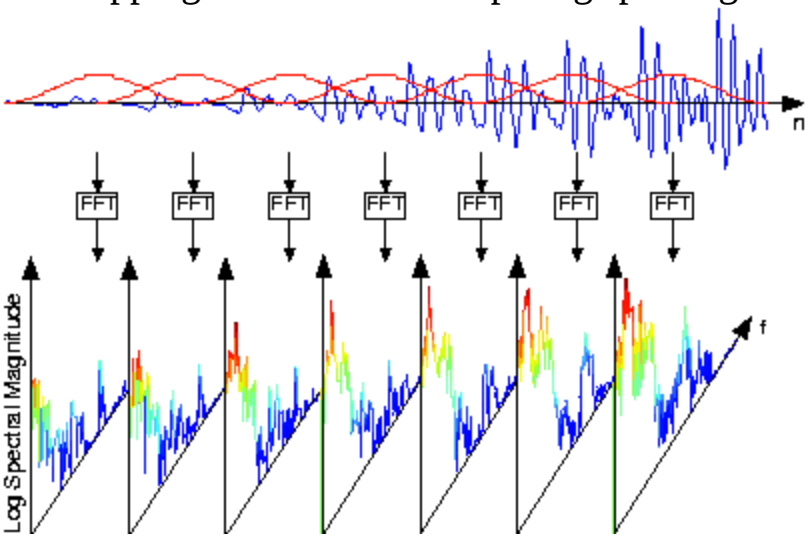


In comparison with the original speech segment shown in the upper plot, the non-overlapped Hanning windowed version shown below it is very ragged. Clearly, spectral information extracted from the bottom plot could well miss important features present in the original.

If you examine the windowed signal sections in sequence to examine windowing's effect on signal amplitude, we see that we have managed to amplitude-modulate the signal with the periodically repeated window ([link]). To alleviate this problem, frames are overlapped (typically by half a frame duration). This solution requires more Fourier transform calculations than needed by rectangular windowing, but the spectra are much better behaved and spectral changes are much better captured.

The speech signal, such as shown in the speech spectrogram, is sectioned into overlapping, equal-length frames, with a Hanning window applied to each frame. The spectra of each of these is calculated, and displayed in spectrograms with frequency extending vertically, window time location running horizontally, and spectral magnitude color-coded. [link] illustrates these computations.

Overlapping windows for computing spectrograms



The original speech segment and the sequence of overlapping Hanning windows applied to it are shown in the upper portion. Frames were 256 samples long and a Hanning window was applied with a half-frame overlap. A length-512 FFT of each frame was computed, with the magnitude of the first 257 FFT values displayed vertically, with spectral amplitude values color-coded.

Exercise:

Problem:

Why the specific values of 256 for N and 512 for K? Another issue is how was the length-512 transform of each length-256 windowed frame computed?

Solution:

These numbers are powers-of-two, and the FFT algorithm can be exploited with these lengths. To compute a longer transform than the input signal's duration, we simply zero-pad the signal.

Filtering with the DFT

Introduction

Equation:

$$\begin{aligned} y[n] &= x[n]*h[n] \\ &= \sum_{k=-\infty}^{\infty} x[k]h[n-k] \end{aligned}$$

Equation:

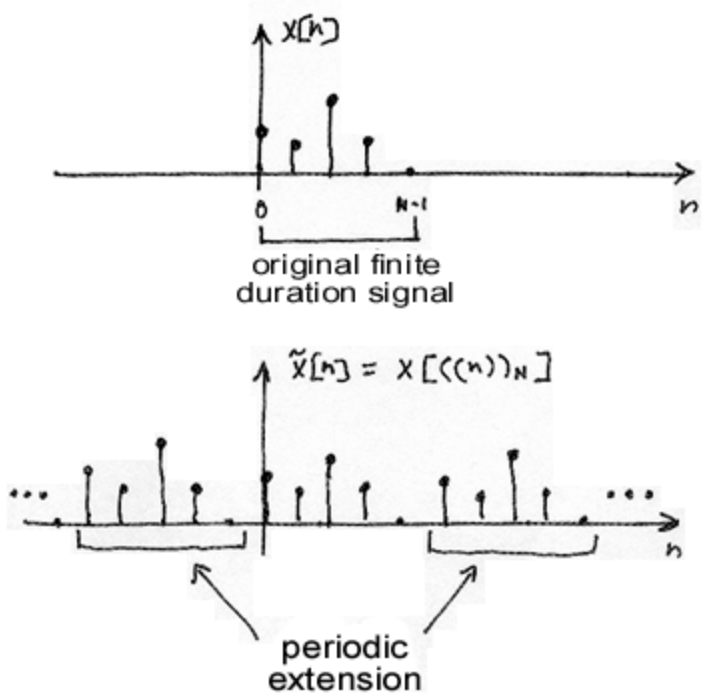
$$Y(\omega) = X(\omega)H(\omega)$$

Assume that $H(\omega)$ is specified.
Exercise:

> **Problem:** How can we implement $X(\omega)H(\omega)$ in a computer?

> **Solution:**
>
> Discretize (sample) $X(\omega)$ and $H(\omega)$. In order to do this, we should take the DFTs of $x[n]$ and $h[n]$ to get $X[k]$ and $X[k]$. Then we will compute
>
> $$\tilde{y}[n] = \text{IDFT}\left(X[k]H[k]\right)$$
>
> Does $\tilde{y}[n] = y[n]$?

Recall that the DFT treats N-point sequences as if they are periodically extended ([[link]]):



Compute IDFT of Y[k]

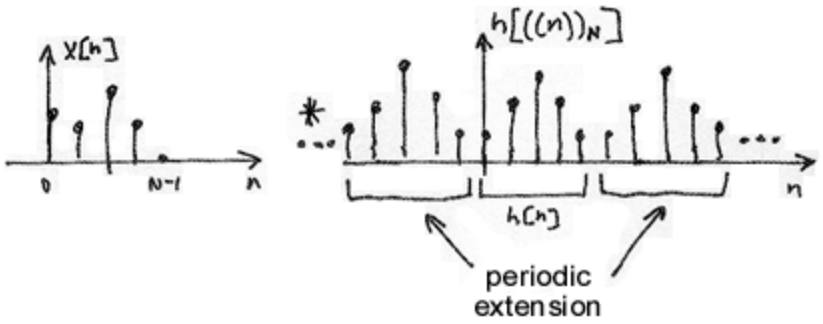
Equation:

$$\begin{aligned}
\tilde{y}[n] &= \tfrac{1}{N}\sum_{k=0}^{N-1} Y[k] e^{i2\pi\frac{k}{N}n} \\
&= \tfrac{1}{N}\sum_{k=0}^{N-1} X[k]H[k] e^{i2\pi\frac{k}{N}n} \\
&= \tfrac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} x[m] e^{-\left(i2\pi\frac{k}{N}m\right)} H[k] e^{i2\pi\frac{k}{N}n} \\
&= \sum_{m=0}^{N-1} x[m]\left(\tfrac{1}{N}\sum_{k=0}^{N-1} H[k] e^{i2\pi\frac{k}{N}(n-m)}\right) \\
&= \sum_{m=0}^{N-1} x[m] h[((n-m))_N]
\end{aligned}$$

And the IDFT periodically extends $h[n]$:

$$\widetilde{h}[n-m] = h[((n-m))_N]$$

This computes as shown in [link]:



Equation:

$$\tilde{y}[n] = \sum_{m=0}^{N-1} x[m]h[((n-m))_N]$$

is called **circular convolution** and is denoted by [link].

$\tilde{y}[n] = x[n] \textcircled{N} h[n]$

The above symbol for the circular convolution is for an N-periodic extension.

DFT Pair

$x[n] \textcircled{N} h[n] \overset{\text{DFT}}{\longleftrightarrow} (X[k])(H[k])$

Note that in general:

$x[n] \textcircled{N} h[n] \neq x[n] * h[n]$

Example:
Regular vs. Circular Convolution
To begin with, we are given the following two length-3 signals:

$$x[n] = \{1, 2, 3\}$$

$$h[n] = \{1, 0, 2\}$$

We can zero-pad these signals so that we have the following discrete sequences:

$$x[n] = \{\ldots, 0, 1, 2, 3, 0, \ldots\}$$

$$h[n] = \{\ldots, 0, 1, 0, 2, 0, \ldots\}$$

where $x[0] = 1$ and $h[0] = 1$.

- Regular Convolution:
  Equation:

  $$y[n] = \sum_{m=0}^{2} x[m]h[n-m]$$

  Using the above convolution formula (refer to the link if you need a review of convolution), we can calculate the resulting value for $y[0]$ to $y[4]$. Recall that because we have two length-3 signals, our convolved signal will be length-5.

  - $n = 0$

    $$\{\ldots, 0, 0, 0, 1, 2, 3, 0, \ldots\}$$

    $$\{\ldots, 0, 2, 0, 1, 0, 0, 0, \ldots\}$$

Equation:

$$\begin{aligned} y[0] &= 1 \times 1 + 2 \times 0 + 3 \times 0 \\ &= 1 \end{aligned}$$

- $n = 1$

$$\{\ldots, 0, 0, 1, 2, 3, 0, \ldots\}$$

$$\{\ldots, 0, 2, 0, 1, 0, 0, \ldots\}$$

Equation:

$$\begin{aligned} y[1] &= 1 \times 0 + 2 \times 1 + 3 \times 0 \\ &= 2 \end{aligned}$$

- $n = 2$

$$\{\ldots, 0, 1, 2, 3, 0, \ldots\}$$

$$\{\ldots, 0, 2, 0, 1, 0, \ldots\}$$

Equation:

$$\begin{aligned} y[2] &= 1 \times 2 + 2 \times 0 + 3 \times 1 \\ &= 5 \end{aligned}$$

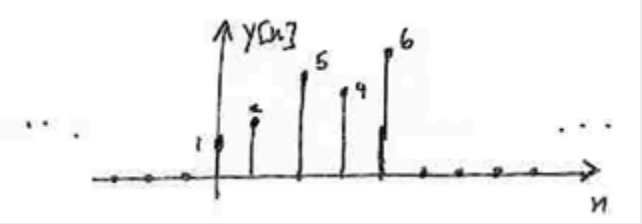
- $n = 3$
  Equation:

$$y[3] = 4$$

- $n = 4$
  Equation:

$$y[4] = 6$$

Regular Convolution Result



Result is finite duration, not periodic!

- Circular Convolution:
  Equation:

$$\tilde{y}[n] = \sum_{m=0}^{2} x[m] h[((n - m))_N]$$

  And now with circular convolution our $h[n]$ changes and becomes a periodically extended signal:
  Equation:

$$h[((n))_N] = \{\ldots, 1, 0, 2, 1, 0, 2, 1, 0, 2, \ldots\}$$

  - $n = 0$

$$\{\ldots, 0, 0, 0, 1, 2, 3, 0, \ldots\}$$

$$\{\ldots, 1, 2, 0, 1, 2, 0, 1, \ldots\}$$

    Equation:

$$\begin{aligned} \tilde{y}[0] &= 1 \times 1 + 2 \times 2 + 3 \times 0 \\ &= 5 \end{aligned}$$

  - $n = 1$

$$\{\ldots, 0, 0, 0, 1, 2, 3, 0, \ldots\}$$

$$\{\ldots, 0, 1, 2, 0, 1, 2, 0, \ldots\}$$

Equation:

$$\begin{aligned} \tilde{y}[1] &= 1 \times 1 + 2 \times 1 + 3 \times 2 \\ &= 8 \end{aligned}$$

- $n = 2$
  Equation:

$$\tilde{y}[2] = 5$$

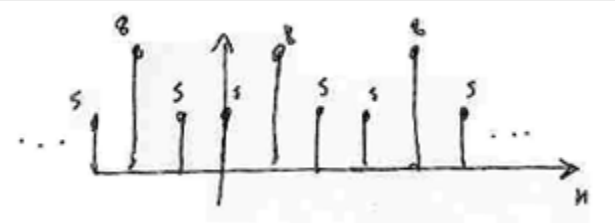
- $n = 3$
  Equation:

$$\tilde{y}[3] = 5$$

- $n = 4$
  Equation:

$$\tilde{y}[4] = 8$$

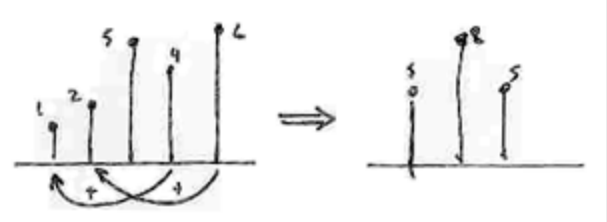
Circular Convolution Result



Result is 3-periodic.

[link] illustrates the relationship between circular convolution and regular convolution using the previous two figures:

Circular Convolution from Regular



The left plot (the circular convolution results) has a "wrap-around" effect due to periodic extension.

Regular Convolution from Periodic Convolution

1. "Zero-pad" $x[n]$ and $h[n]$ to avoid the overlap (wrap-around) effect. We will zero-pad the two signals to a length-5 signal (5 being the duration of the regular convolution result):

$$x[n] = \{1, 2, 3, 0, 0\}$$

$$h[n] = \{1, 0, 2, 0, 0\}$$

2. Now take the DFTs of the zero-padded signals:
Equation:

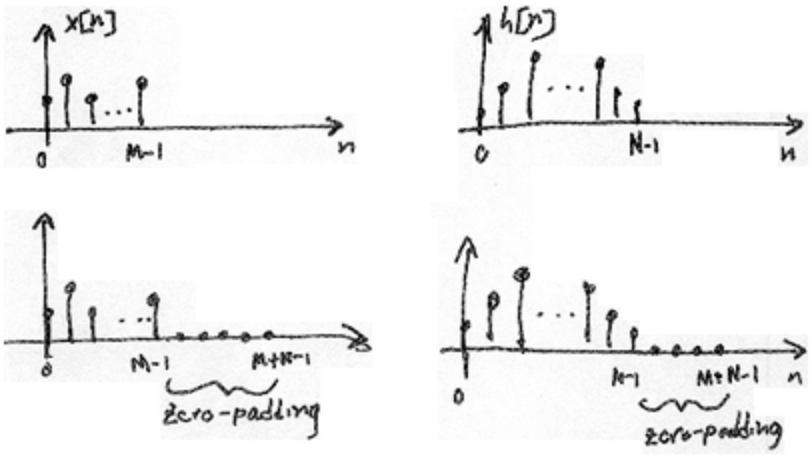
$$\begin{aligned} \tilde{y}[n] &= \frac{1}{N} \sum_{k=0}^{4} X[k] H[k] e^{i 2\pi \frac{k}{5} n} \\ &= \sum_{m=0}^{4} x[m] h[((n - m))_5] \end{aligned}$$

Now we can plot this result ([[link]]):

No Image.

The sequence from 0 to 4 (the underlined part of the sequence) is the regular convolution result. From this illustration we can see that it is 5-periodic!

Note: We can compute the regular convolution result of a convolution of an M-point signal $x[n]$ with an N-point signal $h[n]$ by padding each signal with zeros to obtain two $M+N-1$ length sequences and computing the circular convolution (or equivalently computing the IDFT of $H[k]X[k]$, the product of the DFTs of the zero-padded signals) ([link]).



Note that the lower two images are simply the top images that have been zero-padded.

DSP System

No Image.

The system has a length N impulse response, $h[n]$

1. Sample finite duration continuous-time input $x(t)$ to get $x[n]$ where $n = \{0, \ldots, M-1\}$.
2. Zero-pad $x[n]$ and $h[n]$ to length $M + N - 1$.
3. Compute DFTs $X[k]$ and $H[k]$
4. Compute IDFTs of $X[k]H[k]$

$$y[n] = \tilde{y}[n]$$

where $n = \{0, \ldots, M + N - 1\}$.

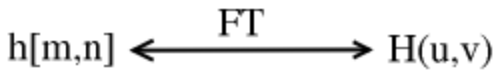
5. Reconstruct $y(t)$

Image Restoration Basics

Image Restoration

In many applications (e.g., satellite imaging, medical imaging, astronomical imaging, poor-quality family portraits) the imaging system introduces a slight distortion. Often images are slightly blurred and image restoration aims at **deblurring** the image.

The blurring can usually be modeled as an LSI system with a given PSF $h[m,n]$.



Fourier Transform (FT) relationship between the two functions.

The observed image is
Equation:

$$g[m,n] = h[m,n] * f[m,n]$$

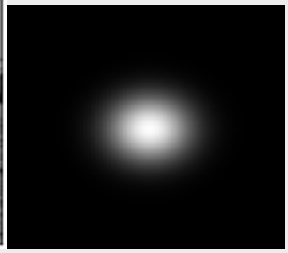
Equation:

$$G(u,v) = H(u,v)F(u,v)$$

Equation:

$$F(u,v) = \frac{G(u,v)}{H(u,v)}$$

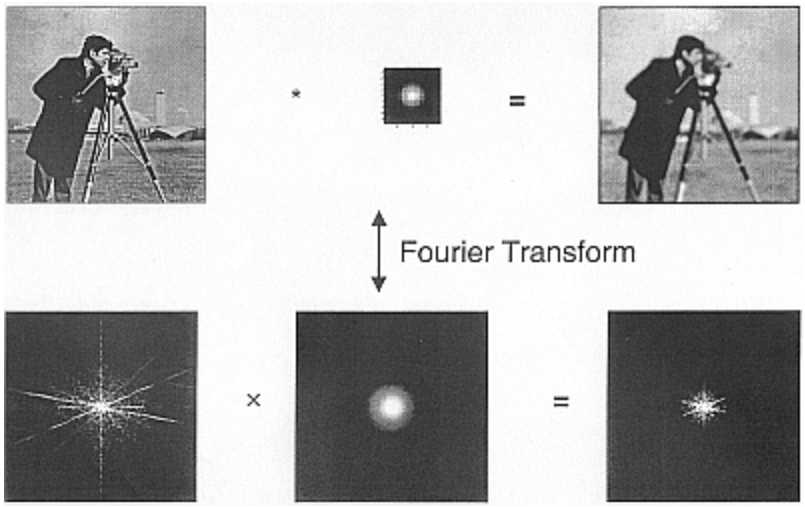
Example:
Image Blurring
Above we showed the equations for representing the common model for blurring an image. In [link] we have an original image and a PSF function that we wish to apply to the image in order to model a basic blurred image.

Once we apply the PSF to the original image, we receive our blurred image that is shown in [link]:

Frequency Domain Analysis

[link] looks at the original images in its typical form; however, it is often useful to look at our images and PSF in the frequency domain. In [link], we take another look at the image blurring example above and look at how the images and results would appear in the frequency domain if we applied the fourier transforms.


*
=
Fourier Transform
×
=

Difference Equation
(Blank Abstract)

Introduction

One of the most important concepts of DSP is to be able to properly represent the input/output relationship to a given LTI system. A linear constant-coefficient **difference equation** (LCCDE) serves as a way to express just this relationship in a discrete-time system. Writing the sequence of inputs and outputs, which represent the characteristics of the LTI system, as a difference equation help in understanding and manipulating a system.

difference equation
: An equation that shows the relationship between consecutive values of a sequence and the differences among them. They are often rearranged as a recursive formula so that a systems output can be computed from the input signal and past outputs.

Example:
Equation:

$$y[n] + 7y[n-1] + 2y[n-2] = x[n] - 4x[n-1]$$

General Formulas for the Difference Equation

As stated briefly in the definition above, a difference equation is a very useful tool in describing and calculating the output of the system described by the formula for a given sample n. The key property of the difference equation is its ability to help easily find the transform, $H(z)$, of a system. In the following two subsections, we will look at the general form of the difference equation and the general conversion to a z-transform directly from the difference equation.

Difference Equation

The general form of a linear, constant-coefficient difference equation (LCCDE), is shown below:
Equation:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

We can also write the general form to easily express a recursive output, which looks like this:
Equation:

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k]$$

From this equation, note that $y[n-k]$ represents the outputs and $x[n-k]$ represents the inputs. The value of N represents the **order** of the difference equation and corresponds to the memory of the system being represented. Because this equation relies on past values of the output, in order to compute a numerical solution, certain past outputs, referred to as the **initial conditions**, must be known.

Conversion to Z-Transform

Using the above formula, [link], we can easily generalize the **transfer function**, $H(z)$, for any difference equation. Below are the steps taken to convert any difference equation into its transfer function, i.e. z-transform. The first step involves taking the Fourier Transform of all the terms in [link]. Then we use the linearity property to pull the transform inside the summation and the time-shifting property of the z-transform to change the time-shifting terms to exponentials. Once this is done, we arrive at the following equation: $a_0 = 1$.
Equation:

$$Y(z) = -\sum_{k=1}^{N} a_k Y(z) z^{-k} + \sum_{k=0}^{M} b_k X(z) z^{-k}$$

Equation:

$$\begin{aligned} H(z) &= \frac{Y(z)}{X(z)} \\ &= \frac{\sum_{k=0}^{M} b_k z^{-k}}{1+\sum_{k=1}^{N} a_k z^{-k}} \end{aligned}$$

Conversion to Frequency Response

Once the z-transform has been calculated from the difference equation, we can go one step further to define the frequency response of the system, or filter, that is being represented by the difference equation.

Note: Remember that the reason we are dealing with these formulas is to be able to aid us in filter design. A LCCDE is one of the easiest ways to represent FIR filters. By being able to find the frequency response, we will be able to look at the basic properties of any filter represented by a simple LCCDE.

Below is the general formula for the frequency response of a z-transform. The conversion is simple a matter of taking the z-transform formula, $H(z)$, and replacing every instance of z with e^{iw}.
Equation:

$$\begin{aligned} H(w) &= H(z)|_{z,z=e^{iw}} \\ &= \frac{\sum_{k=0}^{M} b_k e^{-(iwk)}}{\sum_{k=0}^{N} a_k e^{-(iwk)}} \end{aligned}$$

Once you understand the derivation of this formula, look at the module concerning Filter Design from the Z-Transform for a look into how all of these ideas of the Z-transform, Difference Equation, and Pole/Zero Plots play a role in filter design.

Example

Example:
Finding Difference Equation
Below is a basic example showing the opposite of the steps above: given a transfer function one can easily calculate the systems difference equation.
Equation:

$$H(z) = \frac{(z+1)^2}{\left(z-\frac{1}{2}\right)\left(z+\frac{3}{4}\right)}$$

Given this transfer function of a time-domain filter, we want to find the difference equation. To begin with, expand both polynomials and divide them by the highest order z.
Equation:

$$\begin{aligned} H(z) &= \frac{(z+1)(z+1)}{\left(z-\frac{1}{2}\right)\left(z+\frac{3}{4}\right)} \\ &= \frac{z^2+2z+1}{z^2+2z+1-\frac{3}{8}} \\ &= \frac{1+2z^{-1}+z^{-2}}{1+\frac{1}{4}z^{-1}-\frac{3}{8}z^{-2}} \end{aligned}$$

From this transfer function, the coefficients of the two polynomials will be our a_k and b_k values found in the general difference equation formula, [link]. Using these coefficients and the above form of the transfer function, we can easily write the difference equation:
Equation:

$$x[n] + 2x[n-1] + x[n-2] = y[n] + \frac{1}{4}y[n-1] - \frac{3}{8}y[n-2]$$

In our final step, we can rewrite the difference equation in its more common form showing the recursive nature of the system.
Equation:

$$y[n] = x[n] + 2x[n-1] + x[n-2] + \frac{-1}{4}y[n-1] + \frac{3}{8}y[n-2]$$

Solving a LCCDE

In order for a linear constant-coefficient difference equation to be useful in analyzing a LTI system, we must be able to find the systems output based upon a known input, $x(n)$, and a set of initial conditions. Two common methods exist for solving a LCCDE: the **direct method** and the **indirect method**, the later being based on the z-transform. Below we will briefly discuss the formulas for solving a LCCDE using each of these methods.

Direct Method

The final solution to the output based on the direct method is the sum of two parts, expressed in the following equation:
Equation:

$$y(n) = y_h(n) + y_p(n)$$

The first part, $y_h(n)$, is referred to as the **homogeneous solution** and the second part, $y_h(n)$, is referred to as **particular solution**. The following method is very similar to that used to solve many differential equations, so if you have taken a differential calculus course or used differential equations before then this should seem very familiar.

Homogeneous Solution

We begin by assuming that the input is zero, $x(n) = 0$. Now we simply need to solve the homogeneous difference equation:
Equation:

$$\sum_{k=0}^{N} a_k y[n-k] = 0$$

In order to solve this, we will make the assumption that the solution is in the form of an exponential. We will use lambda, λ, to represent our exponential terms. We now have to solve the following equation:
Equation:

$$\sum_{k=0}^{N} a_k \lambda^{n-k} = 0$$

We can expand this equation out and factor out all of the lambda terms. This will give us a large polynomial in parenthesis, which is referred to as the **characteristic polynomial**. The roots of this polynomial will be the key to solving the homogeneous equation. If there are all distinct roots, then the general solution to the equation will be as follows:
Equation:

$$y_h(n) = C_1(\lambda_1)^n + C_2(\lambda_2)^n + \ldots + C_N(\lambda_N)^n$$

However, if the characteristic equation contains multiple roots then the above general solution will be slightly different. Below we have the modified version for an equation where λ_1 has K multiple roots:
Equation:

$$y_h(n) = C_1(\lambda_1)^n + C_1 n(\lambda_1)^n + C_1 n^2(\lambda_1)^n + \ldots + C_1 n^{K-1}(\lambda_1)^n + C_2(\lambda_2)^n + \ldots + C_N(\lambda_N)^n$$

Particular Solution

The particular solution, $y_p(n)$, will be any solution that will solve the general difference equation:
Equation:

$$\sum_{k=0}^{N} a_k y_p(n-k) = \sum_{k=0}^{M} b_k x(n-k)$$

In order to solve, our guess for the solution to $y_p(n)$ will take on the form of the input, $x(n)$. After guessing at a solution to the above equation involving the particular solution, one only needs to plug the solution into the difference equation and solve it out.

Indirect Method

The indirect method utilizes the relationship between the difference equation and z-transform, discussed earlier, to find a solution. The basic idea is to convert the difference equation into a z-transform, as described above, to get the resulting output, $Y(z)$. Then by inverse transforming this and using partial-fraction expansion, we can arrive at the solution.
Equation:

$$Z\{y\,(n+1) - y\,(n)\} = zY\,(z) - y\,(0)$$

This can be interatively extended to an arbitrary order derivative as in Equation [link].
Equation:

$$Z\left\{ -\sum_{m=0}^{N-1} y\,(n\text{-}m) \right\} = z^n Y\,(z) - \sum_{m=0}^{N-1} z^{n-m-1} y^{(m)}\,(0)$$

Now, the Laplace transform of each side of the differential equation can be taken
Equation:

$$Z\left\{\sum_{k=0}^{N} a_k \left[y\left(n\text{-}m\text{+}1\right) - \sum_{m=0}^{N-1} y\left(n\text{-}m\right) y\left(n\right) \right] = Z\{x(n)\} \right\}$$

which by linearity results in
Equation:

$$\sum_{k=0}^{N} a_k Z\left\{ y\left(n\text{-}m\text{+}1\right) - \sum_{m=0}^{N-1} y\left(n\text{-}m\right) y\left(n\right) \right\} = Z\{x(n)\}$$

and by differentiation properties in
Equation:

$$\sum_{k=0}^{N} a_k \left(z^k Z\{y(n)\} - \sum_{m=0}^{N-1} z^{k-m-1} y^{(m)}\left(0\right) \right) = Z\{x(n)\}.$$

Rearranging terms to isolate the Laplace transform of the output,
Equation:

$$Z\{y(n)\} = \frac{Z\{x(n)\} + \sum_{k=0}^{N} \sum_{m=0}^{k-1} a_k z^{k-m-1} y^{(m)}\left(0\right)}{\sum_{k=0}^{N} a_k z^k}.$$

Thus, it is found that
Equation:

$$Y\left(z\right) = \frac{X\left(z\right) + \sum_{k=0}^{N} \sum_{m=0}^{k-1} a_k z^{k-m-1} y^{(m)}\left(0\right)}{\sum_{k=0}^{N} a_k z^k}.$$

In order to find the output, it only remains to find the Laplace transform $X(z)$ of the input, substitute the initial conditions, and compute the inverse Z-transform of the result. Partial fraction expansions are often required for this last step. This may sound daunting while looking at [link], but it is often easy in practice, especially for low order difference equations. [link] can also be used to determine the transfer function and frequency response.

As an example, consider the difference equation
Equation:

$$y\left[n\text{-}2\right] + 4y\left[n\text{-}1\right] + 3y\left[n\right] = \cos\left(n\right)$$

with the initial conditions $y'(0) = 1$ and $y(0) = 0$ Using the method described above, the Z transform of the solution $y[n]$ is given by
Equation:

$$Y[z] = \frac{z}{[z^2+1][z+1][z+3]} + \frac{1}{[z+1][z+3]}.$$

Performing a partial fraction decomposition, this also equals
Equation:

$$Y[z] = .25\frac{1}{z+1} - .35\frac{1}{z+3} + .1\frac{z}{z^2+1} + .2\frac{1}{z^2+1}.$$

Computing the inverse Laplace transform,
Equation:

$$y(n) = \left(.25z^{-n} - .35z^{-3n} + .1\cos(n) + .2\sin(n)\right)u(n).$$

One can check that this satisfies that this satisfies both the differential equation and the initial conditions.

The Z Transform: Definition
A brief definition of the z-transform, explaining its relationship with the Fourier transform and its region of convergence, ROC.

Basic Definition of the Z-Transform

The **z-transform** of a sequence is defined as
Equation:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}$$

Sometimes this equation is referred to as the **bilateral z-transform**. At times the z-transform is defined as
Equation:

$$X(z) = \sum_{n=0}^{\infty} x[n] z^{-n}$$

which is known as the **unilateral z-transform**.

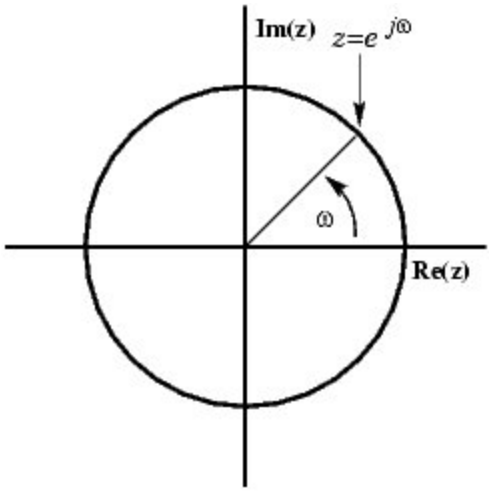
There is a close relationship between the z-transform and the **Fourier transform** of a discrete time signal, which is defined as
Equation:

$$X\left(e^{i\omega}\right) = \sum_{n=-\infty}^{\infty} x[n] e^{-(i\omega n)}$$

Notice that that when the z^{-n} is replaced with $e^{-(i\omega n)}$ the z-transform reduces to the Fourier Transform. When the Fourier Transform exists, $z = e^{i\omega}$, which is to have the magnitude of z equal to unity.

The Complex Plane

In order to get further insight into the relationship between the Fourier Transform and the Z-Transform it is useful to look at the complex plane or **z-plane**. Take a look at the complex plane:
Z-Plane



The Z-plane is a complex plane with an imaginary and real axis referring to the complex-valued variable z. The position on the complex plane is given by $re^{i\omega}$, and the angle from the positive, real axis around the plane is denoted by ω. $X(z)$ is defined everywhere on this plane. $X\left(e^{i\omega}\right)$ on the other hand is defined only where $|z| = 1$, which is referred to as the unit circle. So for example, $\omega = 1$ at $z = 1$ and $\omega = \pi$ at $z = -1$. This is useful because, by representing the Fourier transform as the z-transform on the unit circle, the periodicity of Fourier transform is easily seen.

Region of Convergence

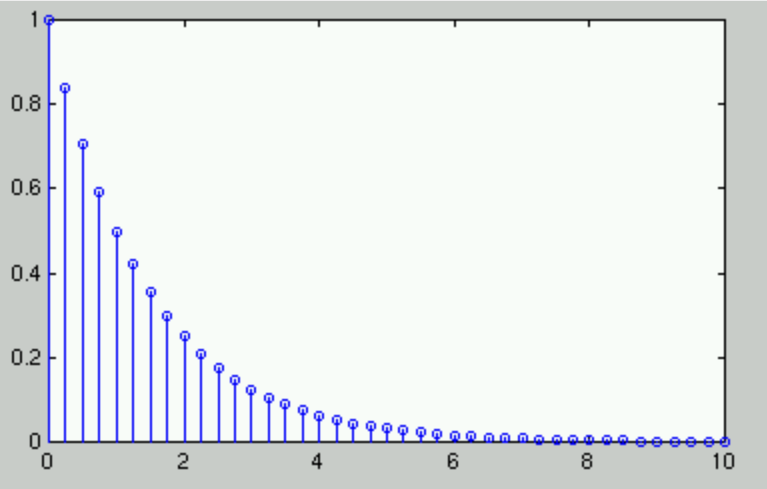
The region of convergence, known as the **ROC**, is important to understand because it defines the region where the z-transform exists. The ROC for a given $x[n]$, is defined as the range of z for which the z-transform converges. Since the z-transform is a **power series**, it converges when $x[n]z^{-n}$ is absolutely summable. Stated differently,
Equation:

$$\sum_{n=-\infty}^{\infty} \left|x[n]z^{-n}\right| < \infty$$

must be satisfied for convergence. This is best illustrated by looking at the different ROC's of the z-transforms of $\alpha^n u[n]$ and $\alpha^n u[n-1]$.

Example:
For
Equation:

$$x[n] = \alpha^n u[n]$$



$x[n] = \alpha^n u[n]$ where $\alpha = 0.5$.

Equation:

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x[n] z^{-n} \\ &= \sum_{n=-\infty}^{\infty} \alpha^n u[n] z^{-n} \\ &= \sum_{n=0}^{\infty} \alpha^n z^{-n} \\ &= \sum_{n=0}^{\infty} \left(\alpha z^{-1}\right)^n \end{aligned}$$

This sequence is an example of a right-sided exponential sequence because it is nonzero for $n \geq 0$. It only converges when $\left|\alpha z^{-1}\right| < 1$. When it converges,

Equation:

$$\begin{aligned} X(z) &= \frac{1}{1-\alpha z^{-1}} \\ &= \frac{z}{z-\alpha} \end{aligned}$$

If $\left|\alpha z^{-1}\right| \geq 1$, then the series, $\sum_{n=0}^{\infty} \left(\alpha z^{-1}\right)^n$ does not converge. Thus the ROC is the range of values where

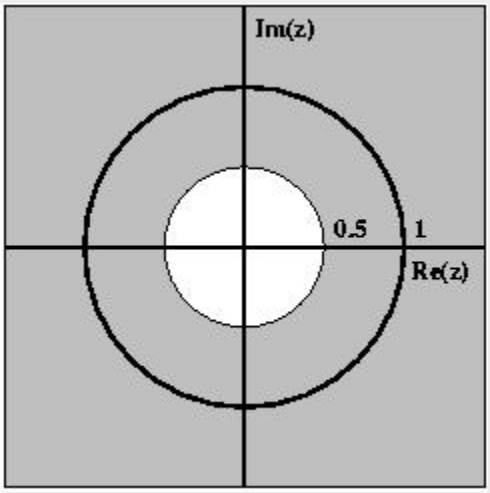
Equation:

$$\left|\alpha z^{-1}\right| < 1$$

or, equivalently,

Equation:

$$|z| > |\alpha|$$



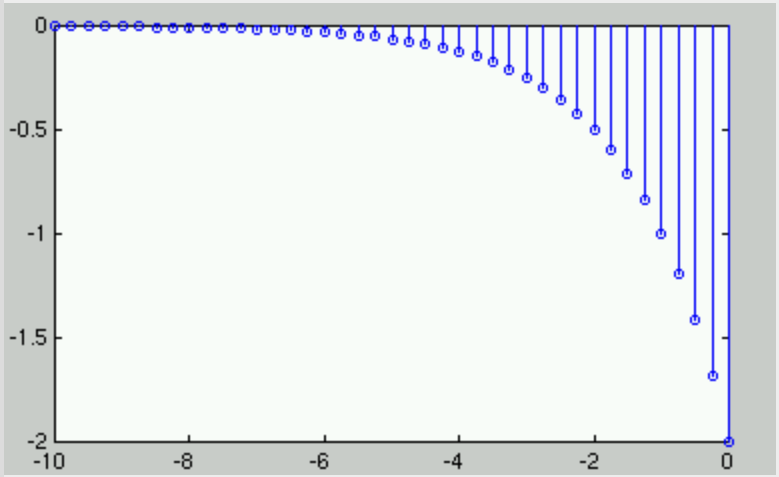
ROC for $x[n] = \alpha^n u[n]$
where $\alpha = 0.5$

Example:

For

Equation:

$$x[n] = (-\alpha^n)u[(-n) - 1]$$



$x[n] = (-\alpha^n)u[(-n) - 1]$ where
$\alpha = 0.5.$

Equation:

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x[n]z^{-n} \\ &= \sum_{n=-\infty}^{\infty} (-\alpha^n)u[\text{-n} - 1]z^{-n} \\ &= -\sum_{n=-\infty}^{-1} \alpha^n z^{-n} \\ &= -\sum_{n=-\infty}^{-1} \left(\alpha^{-1}z\right)^{-n} \\ &= -\sum_{n=1}^{\infty} \left(\alpha^{-1}z\right)^{n} \\ &= 1 - \sum_{n=0}^{\infty} \left(\alpha^{-1}z\right)^{n} \end{aligned}$$

The ROC in this case is the range of values where
Equation:

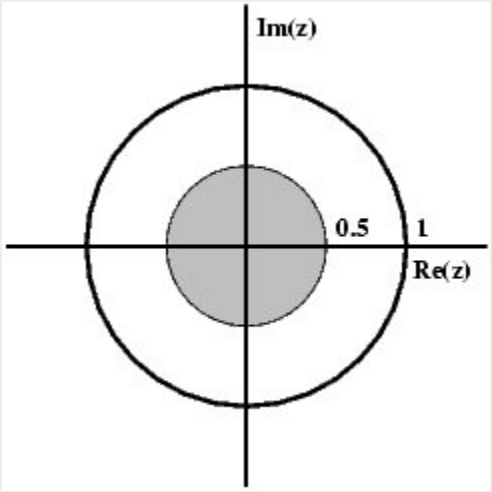
$$\left|\alpha^{-1}z\right| < 1$$

or, equivalently,
Equation:

$$|z| < |\alpha|$$

If the ROC is satisfied, then
Equation:

$$\begin{aligned} X(z) &= 1 - \frac{1}{1-\alpha^{-1}z} \\ &= \frac{z}{z-\alpha} \end{aligned}$$



ROC for
$x[n] = (-\alpha^n)u[(-n) - 1]$

Table of Common z-Transforms
Lists the z-transform and region of convergence (ROC) for several common discrete-time signals.

The table below provides a number of unilateral and bilateral **z-transforms**. The table also specifies the region of convergence.

Note: The notation for z found in the table below may differ from that found in other tables. For example, the basic z-transform of $u[n]$ can be written as either of the following two expressions, which are equivalent:

Equation:

$$\frac{z}{z-1} = \frac{1}{1-z^{-1}}$$

Signal	Z-Transform	ROC
$\delta[n-k]$	z^{-k}	$\mathrm{All}(z)$
$u[n]$	$\frac{z}{z-1}$	$\lvert z\rvert > 1$
$-u[(-n)-1]$	$\frac{z}{z-1}$	$\lvert z\rvert < 1$
$nu[n]$	$\frac{z}{(z-1)^2}$	$\lvert z\rvert > 1$

Signal	Z-Transform	ROC
$n^2u[n]$	$\frac{z(z+1)}{(z-1)^3}$	$\|z\| > 1$
$n^3u[n]$	$\frac{z\left(z^2+4z+1\right)}{(z-1)^4}$	$\|z\| > 1$
$(-\alpha^n)u[(-n)-1]$	$\frac{\mathrm{z}}{z-\alpha}$	$\|z\| < \|\mathrm{\alpha}\|$
$\alpha^nu[n]$	$\frac{z}{z-\alpha}$	$\|z\| > \|\alpha\|$
$n\alpha^nu[n]$	$\frac{\alpha z}{(z-\alpha)^2}$	$\|z\| > \|\alpha\|$
$n^2\alpha^nu[n]$	$\frac{\alpha z(z+\alpha)}{(z-\alpha)^3}$	$\|z\| > \|\alpha\|$
$\frac{\prod_{k=1}^{m} n-k+1}{\alpha^m m!}\alpha^nu[n]$	$\frac{z}{(z-\alpha)^{m+1}}$	
$\gamma^n\cos(\alpha n)u[n]$	$\frac{z(z-\gamma\cos(\alpha))}{z^2-(2\gamma\cos(\alpha))z+\gamma^2}$	$\|z\| > \|\gamma\|$
$\gamma^n\sin(\alpha n)u[n]$	$\frac{z\gamma\sin(\alpha)}{z^2-(2\gamma\cos(\alpha))z+\gamma^2}$	$\|z\| > \|\gamma\|$

Understanding Pole/Zero Plots on the Z-Plane
This module will look at the relationships between the z-transform and the complex plane. Specifically, the creation of pole/zero plots and some of their useful properties are discussed.

Introduction to Poles and Zeros of the Z-Transform

It is quite difficult to qualitatively analyze the Laplace transform and Z-transform, since mappings of their magnitude and phase or real part and imaginary part result in multiple mappings of 2-dimensional surfaces in 3-dimensional space. For this reason, it is very common to examine a plot of a transfer function's poles and zeros to try to gain a qualitative idea of what a system does.

Once the Z-transform of a system has been determined, one can use the information contained in function's polynomials to graphically represent the function and easily observe many defining characteristics. The Z-transform will have the below structure, based on Rational Functions:
Equation:

$$X(z) = \frac{P(z)}{Q(z)}$$

The two polynomials, $P(z)$ and $Q(z)$, allow us to find the poles and zeros of the Z-Transform.

zeros
: The value(s) for z where $P(z) = 0$.
: The complex frequencies that make the overall gain of the filter transfer function zero.

poles
: The value(s) for z where $Q(z) = 0$.
: The complex frequencies that make the overall gain of the filter transfer function infinite.

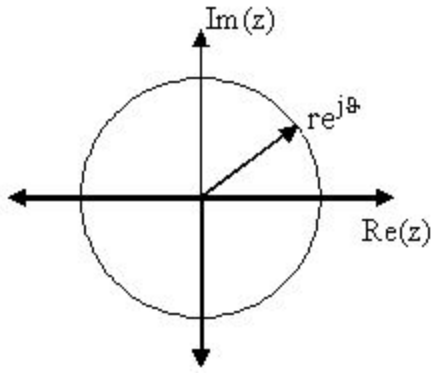
Example:
Below is a simple transfer function with the poles and zeros shown below it.

$$H(z) = \frac{z+1}{\left(z-\frac{1}{2}\right)\left(z+\frac{3}{4}\right)}$$

The zeros are: $\{-1\}$
The poles are: $\left\{\frac{1}{2}, -\frac{3}{4}\right\}$

The Z-Plane

Once the poles and zeros have been found for a given Z-Transform, they can be plotted onto the Z-Plane. The Z-plane is a complex plane with an imaginary and real axis referring to the complex-valued variable z. The position on the complex plane is given by $re^{i\theta}$ and the angle from the positive, real axis around the plane is denoted by θ. When mapping poles and zeros onto the plane, poles are denoted by an "x" and zeros by an "o". The below figure shows the Z-Plane, and examples of plotting zeros and poles onto the plane can be found in the following section.
Z-Plane



Examples of Pole/Zero Plots

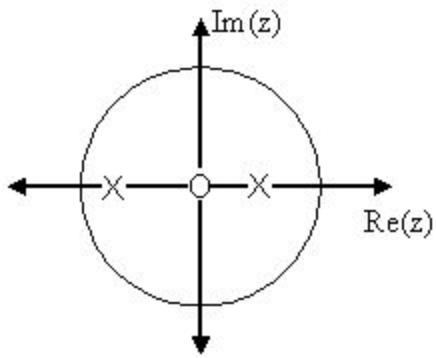
This section lists several examples of finding the poles and zeros of a transfer function and then plotting them onto the Z-Plane.

Example:
Simple Pole/Zero Plot

$$H(z) = \frac{z}{\left(z - \frac{1}{2}\right)\left(z + \frac{3}{4}\right)}$$

The zeros are: $\{0\}$
The poles are: $\left\{\frac{1}{2}, -\frac{3}{4}\right\}$

Pole/Zero Plot



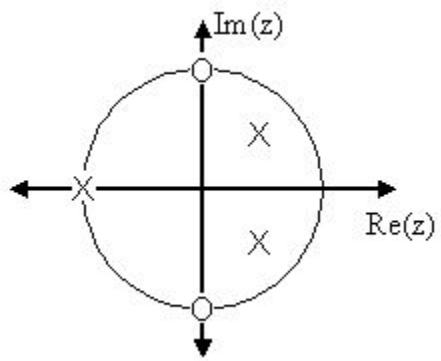
Using the zeros and poles found from the transfer function, the one zero is mapped to zero and the two poles are placed at $\frac{1}{2}$ and $-\frac{3}{4}$

Example:
Complex Pole/Zero Plot

$$H(z) = \frac{(z - i)(z + i)}{\left(z - \left(\frac{1}{2} - \frac{1}{2}i\right)\right)\left(z - \frac{1}{2} + \frac{1}{2}i\right)}$$

The zeros are: $\{i, -i\}$
The poles are: $\left\{-1, \frac{1}{2} + \frac{1}{2}i, \frac{1}{2} - \frac{1}{2}i\right\}$

Pole/Zero Plot



Using the zeros and poles found from the transfer function, the zeros are mapped to $\pm(i)$, and the poles are placed at -1, $\frac{1}{2} + \frac{1}{2}i$ and $\frac{1}{2} - \frac{1}{2}i$

Example:
Pole-Zero Cancellation

An easy mistake to make with regards to poles and zeros is to think that a function like $\frac{(s+3)(s-1)}{s-1}$ is the same as $s + 3$. In theory they are equivalent, as the pole and zero at $s = 1$ cancel each other out in what is known as **pole-zero cancellation**. However, think about what may happen if this were a transfer function of a system that was created with physical circuits. In this case, it is very unlikely that the pole and zero would remain in exactly the same place. A minor temperature change, for instance, could cause one of them to move just slightly. If this were to occur a tremendous amount of volatility is created in that area, since there is a change from infinity at the pole to zero at the zero in a very small range of signals. This

is generally a very bad way to try to eliminate a pole. A much better way is to use **control theory** to move the pole to a better place.

Note:
Repeated Poles and Zeros
It is possible to have more than one pole or zero at any given point. For instance, the discrete-time transfer function $H(z) = z^2$ will have two zeros at the origin and the continuous-time function $H(s) = \frac{1}{s^{25}}$ will have 25 poles at the origin.

MATLAB - If access to MATLAB is readily available, then you can use its functions to easily create pole/zero plots. Below is a short program that plots the poles and zeros from the above example onto the Z-Plane.

```
        % Set up vector for zeros
        z = [j ; -j];

        % Set up vector for poles
        p = [-1 ; .5+.5j ; .5-.5j];

        figure(1);
        zplane(z,p);
        title('Pole/Zero Plot for Complex
Pole/Zero Plot Example');
```

Interactive Demonstration of Poles and Zeros

Interact (when online) with a Mathematica CDF demonstrating Pole/Zero Plots. To Download, right-click and save target as .cdf.

Applications for pole-zero plots

Stability and Control theory

Now that we have found and plotted the poles and zeros, we must ask what it is that this plot gives us. Basically what we can gather from this is that the magnitude of the transfer function will be larger when it is closer to the poles and smaller when it is closer to the zeros. This provides us with a qualitative understanding of what the system does at various frequencies and is crucial to the discussion of stability.

Pole/Zero Plots and the Region of Convergence

The region of convergence (ROC) for $X(z)$ in the complex Z-plane can be determined from the pole/zero plot. Although several regions of convergence may be possible, where each one corresponds to a different impulse response, there are some choices that are more practical. A ROC can be chosen to make the transfer function causal and/or stable depending on the pole/zero plot.

Filter Properties from ROC

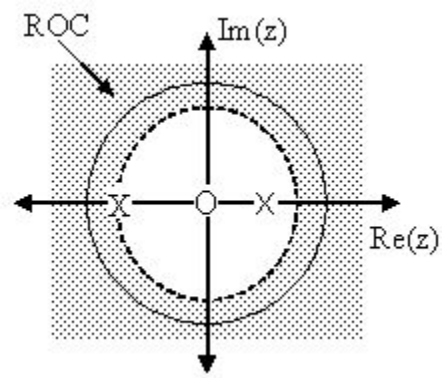
- If the ROC extends outward from the outermost pole, then the system is **causal**.
- If the ROC includes the unit circle, then the system is **stable**.

Below is a pole/zero plot with a possible ROC of the Z-transform in the Simple Pole/Zero Plot discussed earlier. The shaded region indicates the ROC chosen for the filter. From this figure, we can see that the filter will be both causal and stable since the above listed conditions are both met.

Example:

$$H(z) = \frac{z}{\left(z - \frac{1}{2}\right)\left(z + \frac{3}{4}\right)}$$

Region of Convergence for the Pole/Zero Plot



The shaded area represents the chosen ROC for the transfer function.

Frequency Response and Pole/Zero Plots

The reason it is helpful to understand and create these pole/zero plots is due to their ability to help us easily design a filter. Based on the location of the poles and zeros, the magnitude response of the filter can be quickly understood. Also, by starting with the pole/zero plot, one can design a filter and obtain its transfer function very easily.

Filtering in the Frequency Domain
Investigation of different aspects of filtering in the frequency domain, particularly the use of discrete Fourier transforms.

Because we are interested in actual computations rather than analytic calculations, we must consider the details of the discrete Fourier transform. To compute the length-N DFT, we assume that the signal has a duration less than or equal to N. Because frequency responses have an explicit frequency-domain specification in terms of filter coefficients, we don't have a direct handle on which signal has a Fourier transform equaling a given frequency response. Finding this signal is quite easy. First of all, note that the discrete-time Fourier transform of a unit sample equals one for all frequencies. Since the input and output of linear, shift-invariant systems are related to each other by $Y\left(e^{i2\pi f}\right) = H\left(e^{i2\pi f}\right)X\left(e^{i2\pi f}\right)$, **a unit-sample input, which has $X\left(e^{i2\pi f}\right) = 1$, results in the output's Fourier transform equaling the system's transfer function**.
Exercise:

> **Problem:** This statement is a very important result. Derive it yourself.

> **Solution:**
>
> The DTFT of the unit sample equals a constant (equaling 1). Thus, the Fourier transform of the output equals the transfer function.

In the time-domain, the output for a unit-sample input is known as the system's **unit-sample response**, and is denoted by $h(n)$. Combining the frequency-domain and time-domain interpretations of a linear, shift-invariant system's unit-sample response, we have that $h(n)$ and the transfer function are Fourier transform pairs **in terms of the discrete-time Fourier transform**.
Equation:

$$h(n) \leftrightarrow H\left(e^{i2\pi f}\right)$$

Returning to the issue of how to use the DFT to perform filtering, we can analytically specify the frequency response, and derive the corresponding length-N DFT by sampling the frequency response.
Equation:

$$\forall k, k = \{0, \ldots, N-1\} : \left(H(k) = H\left(e^{\frac{i2\pi k}{N}} \right) \right)$$

Computing the inverse DFT yields a length-N signal **no matter what the actual duration of the unit-sample response might be**. If the unit-sample response has a duration less than or equal to N (it's a FIR filter), computing the inverse DFT of the sampled frequency response indeed yields the unit-sample response. If, however, the duration exceeds N, errors are encountered. The nature of these errors is easily explained by appealing to the Sampling Theorem. By sampling in the frequency domain, we have the potential for aliasing in the time domain (sampling in one domain, be it time or frequency, can result in aliasing in the other) unless we sample fast enough. Here, the duration of the unit-sample response determines the minimal sampling rate that prevents aliasing. For FIR systems — they by definition have finite-duration unit sample responses — the number of required DFT samples equals the unit-sample response's duration: $N \geq q$.
Exercise:

Problem:

Derive the minimal DFT length for a length-q unit-sample response using the Sampling Theorem. Because sampling in the frequency domain causes repetitions of the unit-sample response in the time domain, sketch the time-domain result for various choices of the DFT length N.

Solution:

In sampling a discrete-time signal's Fourier transform L times equally over $[0, 2\pi)$ to form the DFT, the corresponding signal equals the periodic repetition of the original signal.
Equation:

$$S(k) \leftrightarrow \sum_{i=-\infty}^{\infty} s(n - iL)$$

To avoid aliasing (in the time domain), the transform length must equal or exceed the signal's duration.

Exercise:

Problem:

Express the unit-sample response of a FIR filter in terms of difference equation coefficients. Note that the corresponding question for IIR filters is far more difficult to answer: Consider the example.

Solution:

The difference equation for an FIR filter has the form
Equation:

$$y(n) = \sum_{m=0}^{q} b_m x(n-m)$$

The unit-sample response equals
Equation:

$$h(n) = \sum_{m=0}^{q} b_m \delta(n-m)$$

which corresponds to the representation described in a problem of a length-q boxcar filter.

For IIR systems, we cannot use the DFT to find the system's unit-sample response: aliasing of the unit-sample response will **always** occur. Consequently, we can only implement an IIR filter accurately in the time

domain with the system's difference equation. **Frequency-domain implementations are restricted to FIR filters.**

Another issue arises in frequency-domain filtering that is related to time-domain aliasing, this time when we consider the output. Assume we have an input signal having duration N_x that we pass through a FIR filter having a length-$q+1$ unit-sample response. What is the duration of the output signal? The difference equation for this filter is

Equation:

$$y(n) = b_0 x(n) + \ldots + b_q x(n-q)$$

This equation says that the output depends on current and past input values, with the input value q samples previous defining the extent of the filter's **memory** of past input values. For example, the output at index N_x depends on $x(N_x)$ (which equals zero), $x(N_x - 1)$, through $x(N_x - q)$. Thus, the output returns to zero only after the last input value passes through the filter's memory. As the input signal's last value occurs at index $N_x - 1$, the last nonzero output value occurs when $n - q = N_x - 1$ or $n = q + N_x - 1$. Thus, the output signal's duration equals $q + N_x$.

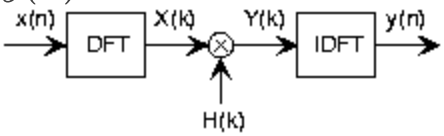
Exercise:

> **Problem:**
>
> In words, we express this result as "The output's duration equals the input's duration plus the filter's duration minus one.". Demonstrate the accuracy of this statement.

> **Solution:**
>
> The unit-sample response's duration is $q+1$ and the signal's N_x. Thus the statement is correct.

The main theme of this result is that a filter's output extends longer than either its input or its unit-sample response. Thus, to avoid aliasing when we use DFTs, the dominant factor is not the duration of input or of the unit-sample response, but of the output. Thus, the number of values at which we

must evaluate the frequency response's DFT must be at least $q + N_x$ and we must compute the same length DFT of the input. To accommodate a shorter signal than DFT length, we simply **zero-pad** the input: Ensure that for indices extending beyond the signal's duration that the signal is zero. Frequency-domain filtering, diagrammed in [link], is accomplished by storing the filter's frequency response as the DFT $H(k)$, computing the input's DFT $X(k)$, multiplying them to create the output's DFT $Y(k) = H(k)X(k)$, and computing the inverse DFT of the result to yield $y(n)$.



To filter a signal in the frequency domain, first compute the DFT of the input, multiply the result by the sampled frequency response, and finally compute the inverse DFT of the product. The DFT's length **must** be at least the sum of the input's and unit-sample response's duration minus one. We calculate these discrete Fourier transforms using the fast Fourier transform algorithm, of course.

Before detailing this procedure, let's clarify why so many new issues arose in trying to develop a frequency-domain implementation of linear filtering.

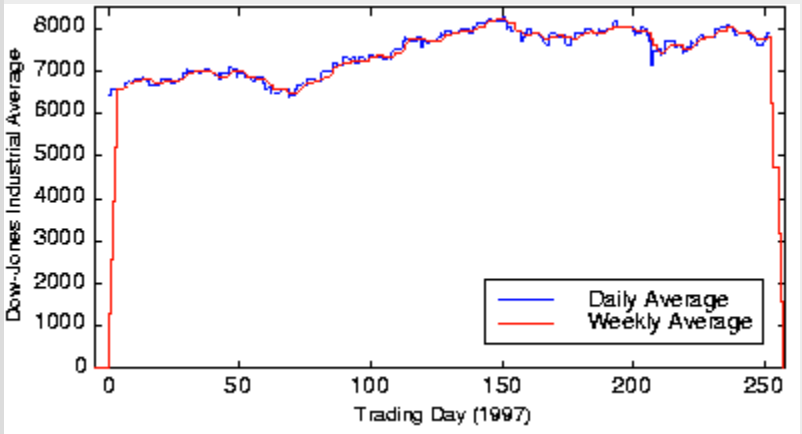
The frequency-domain relationship between a filter's input and output is **always** true: $Y\left(e^{i2\pi f}\right) = H\left(e^{i2\pi f}\right)X\left(e^{i2\pi f}\right)$. The Fourier transforms in this result are discrete-time Fourier transforms; for example, $X\left(e^{i2\pi f}\right) = \sum_n x(n)e^{-(i2\pi fn)}$. Unfortunately, using this relationship to perform filtering is restricted to the situation when we have analytic formulas for the frequency response and the input signal. The reason why we had to "invent" the discrete Fourier transform (DFT) has the same origin: The spectrum resulting from the discrete-time Fourier transform depends on the **continuous** frequency variable f. That's fine for analytic calculation, but computationally we would have to make an uncountably infinite number of computations.

Note: Did you know that two kinds of infinities can be meaningfully defined? A **countably infinite** quantity means that it can be associated with a limiting process associated with integers. An **uncountably infinite** quantity cannot be so associated. The number of rational numbers is countably infinite (the numerator and denominator correspond to locating the rational by row and column; the total number so-located can be counted, voila!); the number of irrational numbers is uncountably infinite. Guess which is "bigger?"

The DFT computes the Fourier transform at a finite set of frequencies — samples the true spectrum — which can lead to aliasing in the time-domain unless we sample sufficiently fast. The sampling interval here is $\frac{1}{K}$ for a length-K DFT: faster sampling to avoid aliasing thus requires a longer transform calculation. Since the longest signal among the input, unit-sample response and output is the output, it is that signal's duration that determines the transform length. We simply extend the other two signals with zeros (zero-pad) to compute their DFTs.

Example:

Suppose we want to average daily stock prices taken over last year to yield a running weekly average (average over five trading sessions). The filter we want is a length-5 averager (as shown in the unit-sample response), and the input's duration is 253 (365 calendar days minus weekend days and holidays). The output duration will be $253 + 5 - 1 = 257$, and this determines the transform length we need to use. Because we want to use the FFT, we are restricted to power-of-two transform lengths. We need to choose any FFT length that exceeds the required DFT length. As it turns out, 256 is a power of two ($2^8 = 256$), and this length just undershoots our required length. To use frequency domain techniques, we must use length-512 fast Fourier transforms.



The blue line shows the Dow Jones Industrial Average from 1997, and the red one the length-5 boxcar-filtered result that provides a running weekly of this market index. Note the "edge" effects in the filtered output.

[link] shows the input and the filtered output. The MATLAB programs that compute the filtered output in the time and frequency domains are

```
Time Domain
h = [1 1 1 1 1]/5;
y = filter(h,1,[djia zeros(1,4)]);
```

```
Frequency Domain
h = [1 1 1 1 1]/5;
DJIA = fft(djia, 512);
H = fft(h, 512);
Y = H.*X;
y = ifft(Y);
```

Note: The `filter` program has the feature that the length of its output equals the length of its input. To force it to produce a signal having the proper length, the program zero-pads the input appropriately.

MATLAB's `fft` function automatically zero-pads its input if the specified transform length (its second argument) exceeds the signal's length. The frequency domain result will have a small imaginary component — largest value is 2.2×10^{-11} — because of the inherent finite precision nature of computer arithmetic. Because of the unfortunate misfit between signal lengths and favored FFT lengths, the number of arithmetic operations in the time-domain implementation is far less than those required by the frequency domain version: 514 versus 62,271. If the input signal had been one sample shorter, the frequency-domain computations would have been more than a factor of two less (28,696), but far more than in the time-domain implementation.

An interesting signal processing aspect of this example is demonstrated at the beginning and end of the output. The ramping up and down that occurs can be traced to assuming the input is zero before it begins and after it ends. The filter "sees" these initial and final values as the difference equation passes over the input. These artifacts can be handled in two ways: we can just ignore the edge effects or the data from previous and succeeding years' last and first week, respectively, can be placed at the ends.

Linear-Phase FIR Filters

THE AMPLITUDE RESPONSE

If the real and imaginary parts of $H^f(\omega)$ are given by
Equation:

$$H^f(\omega) = \Re(\omega) + i\Im(\omega)$$

the magnitude and phase are defined as

$$\left|H^f(\omega)\right| = \sqrt{(\Re(\omega))^2 + (\Im(\omega))^2}$$

$$p(\omega) = \arctan\left(\frac{\Im(\omega)}{\Re(\omega)}\right)$$

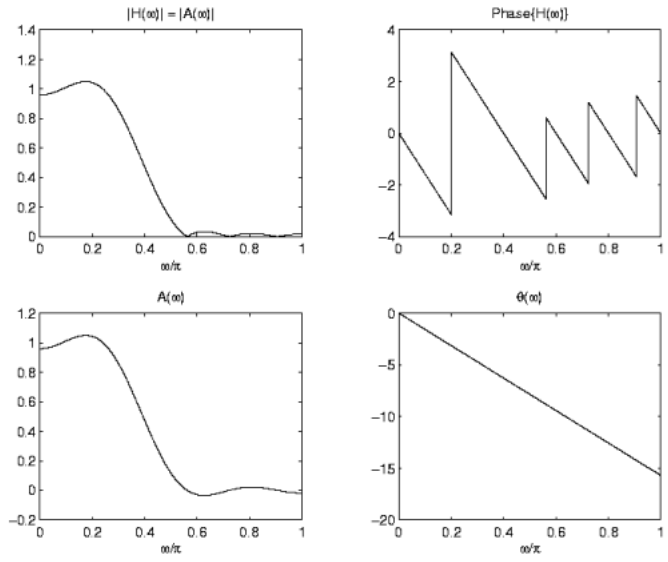
so that
Equation:

$$H^f(\omega) = \left|H^f(\omega)\right| e^{ip(\omega)}$$

With this definition, $\left|H^f(\omega)\right|$ is never negative and $p(\omega)$ is usually discontinuous, but it can be very helpful to write $H^f(\omega)$ as
Equation:

$$H^f(\omega) = A(\omega)e^{i\theta(\omega)}$$

where $A(\omega)$ can be positive and negative, and $\theta(\omega)$ continuous. $A(\omega)$ is called the **amplitude response**. [link] illustrates the difference between $\left|H^f(\omega)\right|$ and $A(\omega)$.



A linear-phase phase filter is one for which the continuous phase $\theta(\omega)$ is linear.

$$H^f(\omega) = A(\omega)e^{i\theta(\omega)}$$

with

$$\theta(\omega) = M\omega + B$$

We assume in the following that the impulse response $h(n)$ is real-valued.

WHY LINEAR-PHASE?

If a discrete-time cosine signal

$$x_1(n) = \cos(\omega_1 n + \varphi_1)$$

is processed through a discrete-time filter with frequency response

$$H^f(\omega) = A(\omega)e^{i\theta(\omega)}$$

then the output signal is given by

$$y_1(n) = A(\omega_1)\cos(\omega_1 n + \varphi_1 + \theta(\omega_1))$$

or

$$y_1(n) = A(\omega_1)\cos\left(\omega_1\left(n + \frac{\theta(\omega_1)}{\omega_1}\right) + \varphi_1\right)$$

The LTI system has the effect of scaling the cosine signal and delaying it by $\frac{\theta(\omega_1)}{\omega_1}$.
Exercise:

> **Problem:** When does the system delay cosine signals with different frequencies by the same amount?

> **Solution:**
>
> - $\frac{\theta(\omega)}{\omega} = \text{constant}$
> - $\theta(\omega) = K\omega$
> - The phase is linear.

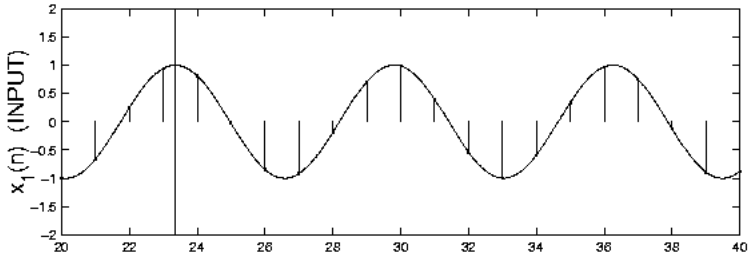
The function $\frac{\theta(\omega)}{\omega}$ is called the **phase delay**. A linear phase filter therefore has constant phase delay.

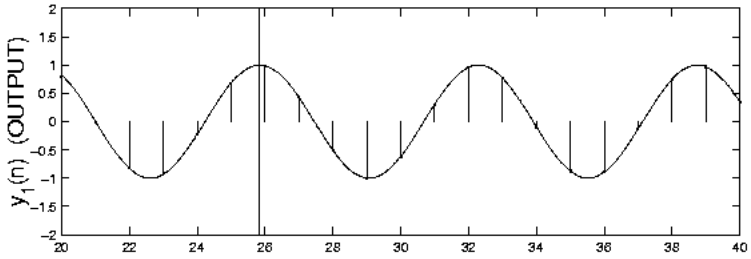
WHY LINEAR-PHASE: EXAMPLE

Consider a discrete-time filter described by the difference equation
Equation:

$$y(n) = 0.1821x(n) + 0.7865x(n-1) - 0.6804x(n-2) + x(n-3) + 0.6804y(n-1) - 0.7865y(n-2) + ($$

When $\omega_1 = 0.31\pi$, then the delay is Math input error. The delay is illustrated in [link]:



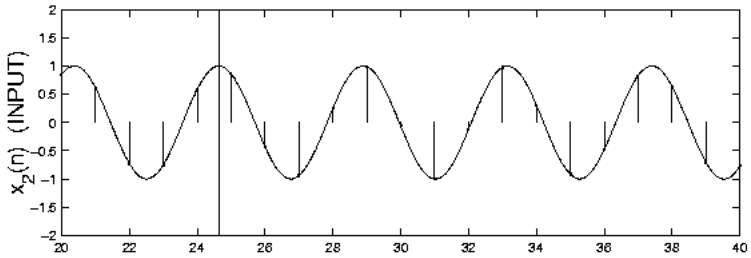


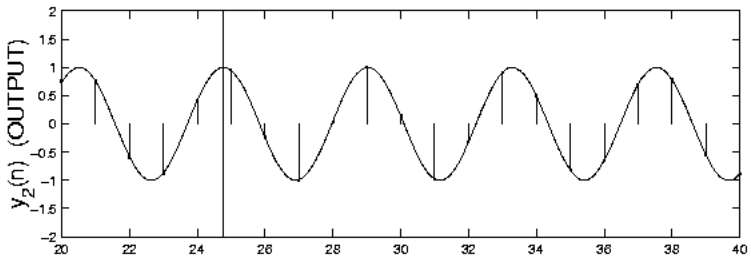
Notice that the delay is fractional --- the discrete-time samples are not exactly reproduced in the output. The fractional delay can be interpreted in this case as a delay of the underlying continuous-time cosine signal.

WHY LINEAR-PHASE: EXAMPLE (2)

Consider the same system given on the previous slide, but let us change the frequency of the cosine signal.

When $\omega_2 = 0.47\pi$, then the delay is Math input error.





Note: For this example, the delay depends on the frequency, because this system does not have linear phase.

WHY LINEAR-PHASE: MORE

From the previous slides, we see that a filter will delay different frequency components of a signal by the same amount if the filter has linear phase (constant phase delay).

In addition, when a narrow band signal (as in AM modulation) goes through a filter, the envelop will be delayed by the **group delay** or **envelop delay** of the filter. The amount by which the envelop is delayed is independent of the carrier frequency only if the filter has linear phase.

Also, in applications like image processing, filters with non-linear phase can introduce artifacts that are visually annoying.

Filter Structures

A realizable filter must require only a finite number of computations per output sample. For linear, causal, time-Invariant filters, this restricts one to rational transfer functions of the form

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}$$

Assuming no pole-zero cancellations, $H(z)$ is FIR if $\forall i, i > 0 : (a_i = 0)$, and IIR otherwise. Filter structures usually implement rational transfer functions as difference equations.

Whether FIR or IIR, a given transfer function can be implemented with many different filter structures. With infinite-precision data, coefficients, and arithmetic, all filter structures implementing the same transfer function produce the same output. However, different filter strucures may produce very different errors with quantized data and finite-precision or fixed-point arithmetic. The computational expense and memory usage may also differ greatly. Knowledge of different filter structures allows DSP engineers to trade off these factors to create the best implementation.

Overview of Digital Filter Design

Advantages of FIR filters

1. Straight forward conceptually and simple to implement
2. Can be implemented with fast convolution
3. Always stable
4. Relatively insensitive to quantization
5. Can have linear phase (same time delay of all frequencies)

Advantages of IIR filters

1. Better for approximating analog systems
2. For a given magnitude response specification, IIR filters often require much less computation than an equivalent FIR, particularly for narrow transition bands

Both FIR and IIR filters are very important in applications.

Generic Filter Design Procedure

1. Choose a desired response, based on application requirements
2. Choose a filter class
3. Choose a quality measure
4. Solve for the filter in class 2 optimizing criterion in 3

Perspective on FIR filtering

Most of the time, people do L optimal design, using the Parks-McClellan algorithm. This is probably the second most important technique in "classical" signal processing (after the Cooley-Tukey (radix-2) FFT).

Most of the time, FIR filters are designed to have linear phase. The most important advantage of FIR filters over IIR filters is that they can have exactly linear phase. There are advanced design techniques for minimum-phase filters, constrained L optimal designs, etc. (see chapter 8 of text). However, if only the **magnitude** of the response is important, IIR filers usually require much fewer operations and are typically used, so the bulk of FIR filter design work has concentrated on linear phase designs.

Window Design Method

The truncate-and-delay design procedure is the simplest and most obvious FIR design procedure.
Exercise:

> **Problem:** Is it any Good?

> **Solution:**
>
> Yes; in fact it's optimal! (in a certain sense)

L2 optimization criterion

find $\forall n, 0 \leq n \leq M-1 : (h[n])$, maximizing the energy difference between the desired response and the actual response: i.e., find

$$\min_{h[n]} \left\{ h[n], \int_{-\pi}^{\pi} \left(|H_d(\omega) - H(\omega)| \right)^2 \, \mathrm{d}\, \omega \right\}$$

by Parseval's relationship
Equation:

$$\begin{aligned} \min_{h[n]} \left\{ h[n], \int_{-\pi}^{\pi} \left(|H_d(\omega) - H(\omega)| \right)^2 \, \mathrm{d}\, \omega \right\} &= 2\pi \sum_{n=-\infty}^{\infty} \left(|h_d[n] - h[n]| \right)^2 \\ &= 2\pi \left(\sum_{n=-\infty}^{-1} \left(|h_d[n] - h[n]| \right)^2 + \sum_{n=0}^{M-1} \left(|h_d[n] - h[n]| \right)^2 + \right. \end{aligned}$$

Since Math input error this becomes

$$\min_{h[n]} \left\{ h[n], \int_{-\pi}^{\pi} \left(|H_d(\omega) - H(\omega)| \right)^2 \, \mathrm{d}\, \omega \right\} = \sum_{h=-\infty}^{-1} \left(|h_d[n]| \right)^2 + \sum_{n=0}^{M-1} \left(|h[n] - h_d[n]| \right)^2 + \sum_{n=M}^{\infty} \left(|h_d[n]| \right)^2$$

Note: $h[n]$ has no influence on the first and last sums.

The best we can do is let

$$h[n] = \begin{cases} h_d[n] \ \text{ if } \ 0 \leq n \leq M-1 \\ 0 \ \text{ if } \ \text{else} \end{cases}$$

Thus $h[n] = h_d[n] w[n]$,

$$w[n] = \begin{cases} 1 \ \text{ if } \ 0 \leq n(M-1) \\ 0 \ \text{ if } \ \text{else} \end{cases}$$

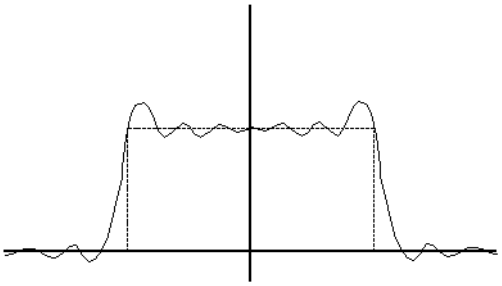
is **optimal** in a least-total-sqaured-error (L_2, or energy) sense!
Exercise:

> **Problem:** Why, then, is this design often considered undersirable?

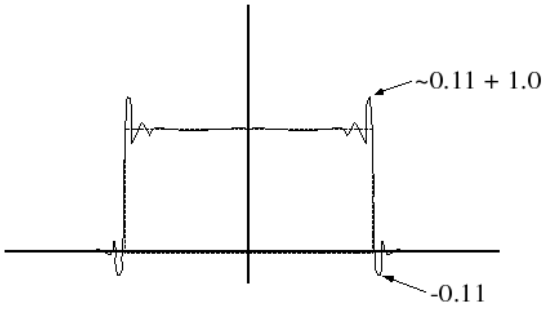
> **Solution:**

Gibbs Phenomenon

$A(\omega)$, small M



$A(\omega)$, large M



For desired spectra with discontinuities, the least-square designs are poor in a minimax (worst-case, or L_∞) error sense.

Window Design Method

Apply a more gradual truncation to reduce "ringing" (Gibb's Phenomenon)

$$\forall n 0 \leq n \leq M-1 h[n] = h_d[n] w[n] : (n 0 \leq n \leq M-1 h[n] = h_d[n] w[n])$$

Note: $H(\omega) = H_d(\omega) * W(\omega)$

The window design procedure (except for the boxcar window) is ad-hoc and not optimal in any usual sense. However, it is very simple, so it is sometimes used for "quick-and-dirty" designs of if the error criterion is itself heurisitic.

Frequency Sampling Design Method for FIR filters

Given a desired frequency response, the frequency sampling design method designs a filter with a frequency response **exactly** equal to the desired response at a particular set of frequencies ω_k.

Equation:

Procedure

$$\forall k, k = [\mathrm{o}, 1, \ldots, N-1] : \left(H_d(\omega_k) = \sum_{n=0}^{M-1} h(n) e^{-(i\omega_k n)} \right)$$

Note: Desired Response must incluce linear phase shift (if linear phase is desired)

Exercise:

Problem: What is $H_d(\omega)$ for an ideal lowpass filter, cotoff at ω_c?

Solution:

$$\begin{cases} e^{-(i\omega \frac{M-1}{2})} \text{ if } -\omega_c \leq \omega \leq \omega_c \\ 0 \text{ if } (-\pi \leq \omega < -\omega_c) \vee (\omega_c < \omega \leq \pi) \end{cases}$$

Note: This set of linear equations can be written in matrix form

Equation:

$$H_d(\omega_k) = \sum_{n=0}^{M-1} h(n) e^{-(i\omega_k n)}$$

Equation:

$$\begin{pmatrix} H_d(\omega_0) \\ H_d(\omega_1) \\ \vdots \\ H_d(\omega_{N-1}) \end{pmatrix} = \begin{pmatrix} e^{-(i\omega_0 0)} & e^{-(i\omega_0 1)} & \dots & e^{-(i\omega_0 (M-1))} \\ e^{-(i\omega_1 0)} & e^{-(i\omega_1 1)} & \dots & e^{-(i\omega_1 (M-1))} \\ \vdots & \vdots & \vdots & \vdots \\ e^{-(i\omega_{M-1} 0)} & e^{-(i\omega_{M-1} 1)} & \dots & e^{-(i\omega_{M-1} (M-1))} \end{pmatrix} \begin{pmatrix} h(0) \\ h(1) \\ \vdots \\ h(M-1) \end{pmatrix}$$

or

$$H_d = W\boldsymbol{h}$$

So
Equation:

$$\boldsymbol{h} = W^{-1} H_d$$

Note: W is a square matrix for $N = M$, and invertible as long as $\omega_i \neq \omega_j + 2\pi l$, $i \neq j$

Important Special Case

What if the frequencies are equally spaced between 0 and 2π, i.e. $\omega_k = \frac{2\pi k}{M} + \alpha$

Then

$$H_d(\omega_k) = \sum_{n=0}^{M-1} h(n) e^{-\left(i\frac{2\pi k n}{M}\right)} e^{-(i\alpha n)} = \sum_{n=0}^{M-1} \left(h(n) e^{-(i\alpha n)}\right) e^{-\left(i\frac{2\pi k n}{M}\right)} = \text{DFT!}$$

so

$$h(n) e^{-(i\alpha n)} = \frac{1}{M} \sum_{k=0}^{M-1} H_d(\omega_k) e^{i\frac{2\pi n k}{M}}$$

or

$$h[n] = \frac{e^{i\alpha n}}{M} \sum_{k=0}^{M-1} H_d[\omega_k] e^{i\frac{2\pi n k}{M}} = e^{i\alpha n}\, \text{IDFT}\left[H_d[\omega_k]\right]$$

Important Special Case #2

$h[n]$ symmetric, linear phase, and has real coefficients. Since $h[n] = h[M-n]$, there are only $\frac{M}{2}$ degrees of freedom, and only $\frac{M}{2}$ linear equations are required.
Equation:

$$\begin{aligned} H[\omega_k] &= \sum_{n=0}^{M-1} h[n] e^{-(i\omega_k n)} \\ &= \begin{cases} \sum_{n=0}^{\frac{M}{2}-1} h[n] \left(e^{-(i\omega_k n)} + e^{-(i\omega_k (M-n-1))} \right) & \text{if M even} \\ \sum_{n=0}^{M-\frac{3}{2}} h[n] \left(e^{-(i\omega_k n)} + e^{-(i\omega_k (M-n-1))} \right) \left(h\left[\frac{M-1}{2}\right] e^{-(i\omega_k \frac{M-1}{2})} \right) & \text{if M odd} \end{cases} \\ &= \begin{cases} e^{-(i\omega_k \frac{M-1}{2})} 2 \sum_{n=0}^{\frac{M}{2}-1} h[n] \cos\left(\omega_k \left(\frac{M-1}{2} - n\right)\right) & \text{if M even} \\ e^{-(i\omega_k \frac{M-1}{2})} 2 \sum_{n=0}^{M-\frac{3}{2}} h[n] \cos\left(\omega_k \left(\frac{M-1}{2} - n\right)\right) + h\left[\frac{M-1}{2}\right] & \text{if M odd} \end{cases} \end{aligned}$$

Removing linear phase from both sides yields

$$A(\omega_k) = \begin{cases} 2 \sum_{n=0}^{\frac{M}{2}-1} h[n] \cos\left(\omega_k \left(\frac{M-1}{2} - n\right)\right) & \text{if M even} \\ 2 \sum_{n=0}^{M-\frac{3}{2}} h[n] \cos\left(\omega_k \left(\frac{M-1}{2} - n\right)\right) + h\left[\frac{M-1}{2}\right] & \text{if M odd} \end{cases}$$

Due to symmetry of response for real coefficients, only $\frac{M}{2}$ ω_k on $\omega \in [0, \pi)$ need be specified, with the frequencies $-\omega_k$ thereby being implicitly defined also. Thus we have $\frac{M}{2}$ **real-valued** simultaneous linear equations to solve for $h[n]$.

Special Case 2a

$h[n]$ symmetric, odd length, linear phase, real coefficients, and ω_k equally spaced:
$\forall k, 0 \leq k \leq M-1 : \left(\omega_k = \frac{n\pi k}{M}\right)$
Equation:

$$\begin{aligned} h[n] &= \text{IDFT}\left[H_d(\omega_k)\right] \\ &= \frac{1}{M} \sum_{k=0}^{M-1} A(\omega_k) e^{-(i\frac{2\pi k}{M})} \frac{M-1}{2} e^{i\frac{2\pi n k}{M}} \\ &= \frac{1}{M} \sum_{k=0}^{M-1} A(k) e^{i(\frac{2\pi k}{M}(n - \frac{M-1}{2}))} \end{aligned}$$

To yield real coefficients, $A(\omega)$ mus be symmetric

$$(A(\omega) = A(-\omega)) \Rightarrow (A[k] = A[M-k])$$

Equation:

$$\begin{aligned} h[n] &= \frac{1}{M}\left(A(0) + \sum_{k=1}^{\frac{M-1}{2}} A[k]\left(e^{i\frac{2\pi k}{M}\left(n-\frac{M-1}{2}\right)} + e^{-\left(i2\pi k\left(n-\frac{M-1}{2}\right)\right)}\right)\right) \\ &= \frac{1}{M}\left(A(0) + 2\sum_{k=1}^{\frac{M-1}{2}} A[k]\cos\left(\frac{2\pi k}{M}\left(n-\frac{M-1}{2}\right)\right)\right) \\ &= \frac{1}{M}\left(A(0) + 2\sum_{k=1}^{\frac{M-1}{2}} A[k](-1)^k\cos\left(\frac{2\pi k}{M}\left(n+\frac{1}{2}\right)\right)\right) \end{aligned}$$

Simlar equations exist for even lengths, anti-symmetric, and $\alpha = \frac{1}{2}$ filter forms.

Comments on frequency-sampled design

This method is simple conceptually and very efficient for equally spaced samples, since $h[n]$ can be computed using the IDFT.

$H(\omega)$ for a frequency sampled design goes **exactly** through the sample points, but it may be very far off from the desired response for $\omega \neq \omega_k$. This is the main problem with frequency sampled design.

Possible solution to this problem: specify more frequency samples than degrees of freedom, and minimize the total error in the frequency response at all of these samples.

Extended frequency sample design

For the samples $H(\omega_k)$ where $0 \leq k \leq M-1$ and $N > M$, find $h[n]$, where $0 \leq n \leq M-1$ minimizing $\parallel H_d(\omega_k) - H(\omega_k) \parallel$

For $\parallel l \parallel_\infty$ norm, this becomes a linear programming problem (standard packages availble!)

Here we will consider the $\parallel l \parallel_2$ norm.

To minimize the $\parallel l \parallel_2$ norm; that is, $\sum_{n=0}^{N-1} |H_d(\omega_k) - H(\omega_k)|$, we have an overdetermined set of linear equations:

$$\begin{pmatrix} e^{-(i\omega_0 0)} & \dots & e^{-(i\omega_0(M-1))} \\ \vdots & \vdots & \vdots \\ e^{-(i\omega_{N-1}0)} & \dots & e^{-(i\omega_{N-1}(M-1))} \end{pmatrix} \boldsymbol{h} = \begin{pmatrix} H_d(\omega_0) \\ H_d(\omega_1) \\ \vdots \\ H_d(\omega_{N-1}) \end{pmatrix}$$

or

$$W\boldsymbol{h} = H_d$$

The minimum error norm solution is well known to be $\boldsymbol{h} = \left(\overline{W}W\right)^{-1}\overline{W}H_d$; $\left(\overline{W}W\right)^{-1}\overline{W}$ is well known as the pseudo-inverse matrix.

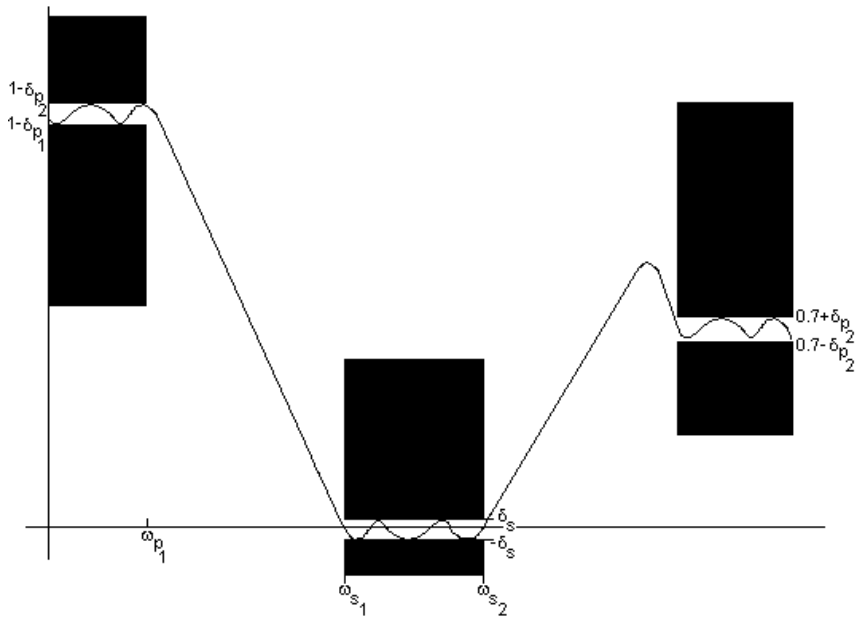
Note: Extended frequency sampled design discourages radical behavior of the frequency response between samples for sufficiently closely spaced samples. However, the actual frequency response may no longer pass exactly through **any** of the $H_d(\omega_k)$.

Parks-McClellan FIR Filter Design

The approximation tolerances for a filter are very often given in terms of the maximum, or worst-case, deviation within frequency bands. For example, we might wish a lowpass filter in a (16-bit) CD player to have no more than $\frac{1}{2}$-bit deviation in the pass and stop bands.

$$H(\omega) = \begin{cases} 1 - \frac{1}{2^{17}} \leq |H(\omega)| \leq 1 + \frac{1}{2^{17}} & \text{if} \quad |\omega| \leq \omega_p \\ \frac{1}{2^{17}} \geq |H(\omega)| & \text{if} \quad \omega_s \leq |\omega| \leq \pi \end{cases}$$

The Parks-McClellan filter design method efficiently designs linear-phase FIR filters that are optimal in terms of worst-case (minimax) error. Typically, we would like to have the shortest-length filter achieving these specifications. Figure [link] illustrates the amplitude frequency response of such a filter.



The black boxes on the left and right are the passbands, the black boxes in the middle represent the stop band, and the space between the boxes are the transition bands. Note that overshoots may be allowed in the transition bands.

Exercise:

Problem: Must there be a transition band?

Solution:

Yes, when the desired response is discontinuous. Since the frequency response of a finite-length filter must be continuous, without a transition band the worst-case error could be no less than half the discontinuity.

Formal Statement of the L-∞ (Minimax) Design Problem

For a given filter length (M) and type (odd length, symmetric, linear phase, for example), and a relative error weighting function $W(\omega)$, find the filter coefficients minimizing the maximum error

$$\underset{\boldsymbol{h}}{\operatorname{argmin}} \underset{\omega \in F}{\operatorname{argmax}} \, |E(\omega)| = \underset{\boldsymbol{h}}{\operatorname{argmin}} \, \| \, E(\omega) \, \|_\infty$$

where

$$E(\omega) = W(\omega) \left(H_d(\omega) - H(\omega) \right)$$

and F is a compact subset of $\omega \in [0, \pi]$ (i.e., all ω in the passbands and stop bands).

Note: Typically, we would often rather specify $\| \, E(\omega) \, \|_\infty \leq \delta$ and minimize over M and $\boldsymbol{h}$; however, the design techniques minimize δ for a given M. One then repeats the design procedure for different M until the minimum M satisfying the requirements is found.

We will discuss in detail the design only of odd-length symmetric linear-phase FIR filters. Even-length and anti-symmetric linear phase FIR filters are essentially the same except for a slightly different implicit weighting function. For arbitrary phase, exactly optimal design procedures have only recently been developed (1990).

Outline of L-∞ Filter Design

The Parks-McClellan method adopts an indirect method for finding the minimax-optimal filter coefficients.

1. Using results from Approximation Theory, simple conditions for determining whether a given filter is L^∞ (minimax) optimal are found.
2. An iterative method for finding a filter which satisfies these conditions (and which is thus optimal) is developed.

That is, the L^∞ filter design problem is actually solved **indirectly**.

Conditions for L-∞ Optimality of a Linear-phase FIR Filter

All conditions are based on Chebyshev's "Alternation Theorem," a mathematical fact from polynomial approximation theory.

Alternation Theorem

Let F be a compact subset on the real axis x, and let $P(x)$ be and Lth-order polynomial

$$P(x) = \sum_{k=0}^{L} a_k x^k$$

Also, let $D(x)$ be a desired function of x that is continuous on F, and $W(x)$ a positive, continuous weighting function on F. Define the error $E(x)$ on F as

$$E(x) = W(x)\left(D(x) - P(x)\right)$$

and

$$\parallel E(x) \parallel_\infty = \operatorname*{argmax}_{x \in F} |E(x)|$$

A necessary and sufficient condition that $P(x)$ is the unique Lth-order polynomial minimizing $\parallel E(x) \parallel_\infty$ is that $E(x)$ exhibits **at least** $L+2$ "alternations;" that is, there must exist at least $L+2$ values of x, $x_k \in F$, $k = [0, 1, \ldots, L+1]$, such that $x_0 < x_1 < \ldots < x_{L+2}$ and such that $E(x_k) = -E(x_{k+1}) = \pm(\parallel E \parallel_\infty)$
Exercise:

Problem: What does this have to do with linear-phase filter design?

Solution:

It's the same problem! To show that, consider an odd-length, symmetric linear phase filter.
Equation:

$$\begin{aligned} H(\omega) &= \textstyle\sum_{n=0}^{M-1} h(n) e^{-(i\omega n)} \\ &= e^{-\left(i\omega\frac{M-1}{2}\right)} \left(h\left(\tfrac{M-1}{2}\right) + 2\textstyle\sum_{n=1}^{L} h\left(\tfrac{M-1}{2} - n\right)\cos(\omega n)\right) \end{aligned}$$

Equation:

$$A(\omega) = h(L) + 2\sum_{n=1}^{L} h(L-n)\cos(\omega n)$$

Where $L \doteq \frac{M-1}{2}$.

Using trigonometric identities (such as
$\cos(n\alpha) = 2\cos((n-1)\alpha)\cos(\alpha) - \cos((n-2)\alpha)$), we can rewrite $A(\omega)$ as

$$A(\omega) = h(L) + 2\sum_{n=1}^{L} h(L-n)\cos(\omega n) = \sum_{k=0}^{L} \alpha_k \cos^k(\omega)$$

where the α_k are related to the $h(n)$ by a linear transformation. Now, let $x = \cos(\omega)$. This is a one-to-one mapping from $x \in [-1, 1]$ onto $\omega \in [0, \pi]$. Thus $A(\omega)$ is an Lth-order polynomial in $x = \cos(\omega)$!

Note: The alternation theorem holds for the L^∞ filter design problem, too!

Therefore, to determine whether or not a length-M, odd-length, symmetric linear-phase filter is optimal in an L^∞ sense, simply count the alternations in $E(\omega) = W(\omega)\left(A_d(\omega) - A(\omega)\right)$ in the pass and stop bands. If there are $L + 2 = \frac{M+3}{2}$ or more alternations, $h(n)$, $0 \leq n \leq M-1$ is the optimal filter!

Optimality Conditions for Even-length Symmetric Linear-phase Filters

For M even,

$$A(\omega) = \sum_{n=0}^{L} h(L-n)\cos\left(\omega\left(n + \frac{1}{2}\right)\right)$$

where $L = \frac{M}{2} - 1$ Using the trigonometric identity $\cos(\alpha+\beta) = \cos(\alpha-\beta) + 2\cos(\alpha)\cos(\beta)$ to pull out the $\frac{\omega}{2}$ term and then using the other trig identities, it can be shown that $A(\omega)$ can be written as

$$A(\omega) = \cos\left(\frac{\omega}{2}\right)\sum_{k=0}^{L} \alpha_k \cos^k(\omega)$$

Again, this is a polynomial in $x = \cos(\omega)$, except for a weighting function out in front.
Equation:

$$\begin{aligned} E(\omega) &= W(\omega)\left(A_d(\omega) - A(\omega)\right) \\ &= W(\omega)\left(A_d(\omega) - \cos\left(\tfrac{\omega}{2}\right)P(\omega)\right) \\ &= W(\omega)\cos\left(\tfrac{\omega}{2}\right)\left(\frac{A_d(\omega)}{\cos\left(\frac{\omega}{2}\right)} - P(\omega)\right) \end{aligned}$$

which implies
Equation:

$$E(x) = W'(x)\left(A'_d(x) - P(x)\right)$$

where

$$W'(x) = W\Big((\cos(x))^{-1}\Big)\cos\left(\frac{1}{2}(\cos(x))^{-1}\right)$$

and

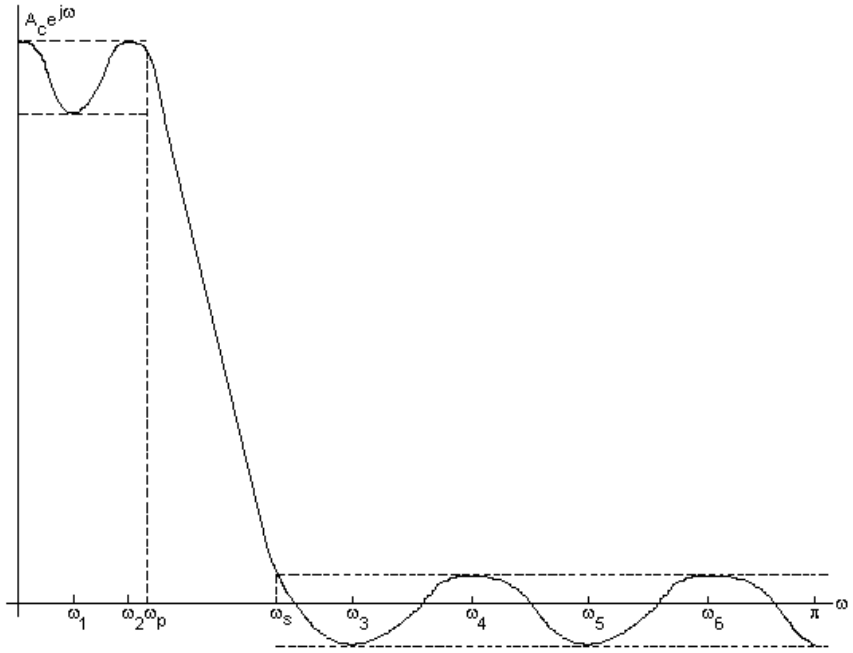
$$A'_d(x) = \frac{A_d\Big((\cos(x))^{-1}\Big)}{\cos\left(\frac{1}{2}(\cos(x))^{-1}\right)}$$

Again, this is a polynomial approximation problem, so the alternation theorem holds. If $E(\omega)$ has at least $L + 2 = \frac{M}{2} + 1$ alternations, the even-length symmetric filter is optimal in an L^∞ sense.

The prototypical filter design problem:

$$W = \begin{cases} 1 \text{ if } & |\omega| \leq \omega_p \\ \frac{\delta_s}{\delta_p} \text{ if } & |\omega_s| \leq |\omega| \end{cases}$$

See [link].



L-∞ Optimal Lowpass Filter Design Lemma

1. The maximum possible number of alternations for a lowpass filter is $L+3$: The proof is that the extrema of a polynomial occur only where the derivative is zero: $\frac{\partial P(x)}{\partial x} = 0$. Since $P'(x)$ is an $(L-1)$th-order polynomial, it can have at **most** $L-1$ zeros. **However**, the mapping $x = \cos(\omega)$ implies that $\frac{\partial A(\omega)}{\partial \omega} = 0$ at $\omega = 0$ and $\omega = \pi$, for two more possible alternation points. **Finally**, the band edges can also be alternations, for a total of $L-1+2+2 = L+3$ possible alternations.
2. There must be an alternation at either $\omega = 0$ or $\omega = \pi$.
3. Alternations must occur at ω_p and ω_s. See [link].
4. The filter must be equiripple except at possibly $\omega = 0$ or $\omega = \pi$. Again see [link].

Note: The alternation theorem doesn't directly suggest a method for computing the optimal filter. It simply tells us how to recognize that a filter **is** optimal, or **isn't** optimal. What we need is an intelligent way of guessing the optimal filter coefficients.

In matrix form, these $L+2$ simultaneous equations become

$$\begin{pmatrix} 1 & \cos(\omega_0) & \cos(2\omega_0) & \dots & \cos(L\omega_0) & \frac{1}{W(\omega_0)} \\ 1 & \cos(\omega_1) & \cos(2\omega_1) & \dots & \cos(L\omega_1) & \frac{-1}{W(\omega_1)} \\ \vdots & \vdots & \ddots & \dots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \dots & \ddots & \vdots \\ 1 & \cos(\omega_{L+1}) & \cos(2\omega_{L+1}) & \dots & \cos(L\omega_{L+1}) & \frac{\pm(1)}{W(\omega_{L+1})} \end{pmatrix} \begin{pmatrix} h(L) \\ h(L-1) \\ \vdots \\ h(1) \\ h(0) \\ \delta \end{pmatrix} = \begin{pmatrix} A_d(\omega_0) \\ A_d(\omega_1) \\ \vdots \\ \vdots \\ \vdots \\ A_d(\omega_{L+1}) \end{pmatrix}$$

or

$$W \begin{pmatrix} \boldsymbol{h} \\ \delta \end{pmatrix} = A_d$$

So, for the given set of $L+2$ extremal frequencies, we can solve for $\boldsymbol{h}$ and δ via $(\boldsymbol{h}\delta)^T = W^{-1}A_d$. Using the FFT, we can compute $A(\omega)$ of $h(n)$, on a dense set of frequencies. If the old ω_k are, in fact the extremal locations of $A(\omega)$, then the alternation theorem is satisfied and $h(n)$ is **optimal**. If not, repeat the process with the new extremal locations.

Computational Cost

$O\left(L^3\right)$ for the matrix inverse and $N \log_2 N$ for the FFT ($N \geq 32L$, typically), **per iteration**!

This method is expensive computationally due to the matrix inverse.

A more efficient variation of this method was developed by Parks and McClellan (1972), and is based on the Remez exchange algorithm. To understand the Remez exchange algorithm, we first need to understand Lagrange Interpoloation.

Now $A(\omega)$ is an Lth-order polynomial in $x = \cos(\omega)$, so Lagrange interpolation can be used to **exactly** compute $A(\omega)$ from $L+1$ samples of $A(\omega_k)$, $k = [0, 1, 2, ..., L]$.

Thus, given a set of extremal frequencies and knowing δ, samples of the amplitude response $A(\omega)$ can be computed **directly** from the
Equation:

$$A(\omega_k) = \frac{(-1)^{k(1)}}{W(\omega_k)}\delta + A_d(\omega_k)$$

without solving for the filter coefficients!

This leads to computational savings!

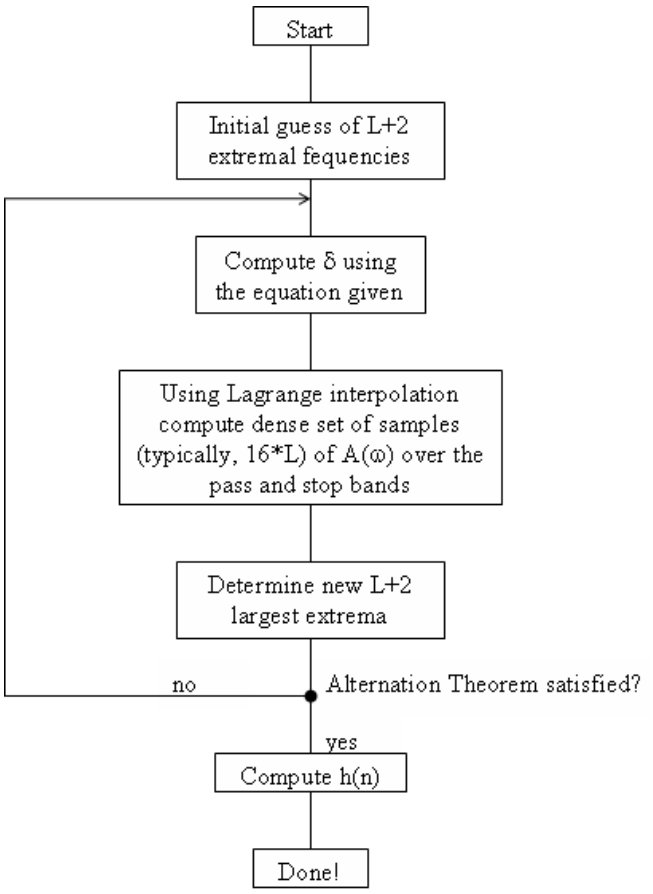
Note that [link] is a set of $L+2$ simultaneous equations, which can be solved for δ to obtain (Rabiner, 1975)
Equation:

$$\delta = \frac{\sum_{k=0}^{L+1} \gamma_k A_d(\omega_k)}{\sum_{k=0}^{L+1} \frac{(-1)^{k(1)}\gamma_k}{W(\omega_k)}}$$

where

$$\gamma_k = \prod_{i=i\neq k,0}^{L+1} \frac{1}{\cos(\omega_k) - \cos(\omega_i)}$$

The result is the Parks-McClellan FIR filter design method, which is simply an application of the Remez exchange algorithm to the filter design problem. See [link].



The initial guess of extremal frequencies is usually equally spaced in the band. Computing δ costs $O\left(L^2\right)$. Using Lagrange interpolation costs $O(16LL) \simeq O\left(16L^2\right)$. Computing $h(n)$ costs $O\left(L^3\right)$, but it is only done once!

The cost per iteration is $O\left(16L^2\right)$, as opposed to $O\left(L^3\right)$; much more efficient for large L. Can also interpolate to DFT sample frequencies, take inverse FFT to get corresponding filter coefficients, and zeropad and take longer FFT to efficiently interpolate.

FIR Filter Design using MATLAB

FIR Filter Design Using MATLAB

Design by windowing

The MATLAB function `fir1()` designs conventional lowpass, highpass, bandpass, and bandstop linear-phase FIR filters based on the windowing method. The command

```
b = fir1(N,Wn)
```

returns in vector `b` the impulse response of a lowpass filter of order `N`. The cut-off frequency `Wn` must be between 0 and 1 with 1 corresponding to the half sampling rate.

The command

```
b = fir1(N,Wn,'high')
```

returns the impulse response of a highpass filter of order `N` with normalized cutoff frequency `Wn`.

Similarly, `b = fir1(N,Wn,'stop')` with `Wn` a two-element vector designating the stopband designs a bandstop filter.

Without explicit specification, the **Hamming window** is employed in the design. Other windowing functions can be used by specifying the windowing function as an extra argument of the function. For example,

Blackman window can be used instead by the command `b = fir1(N, Wn, blackman(N))`.

Parks-McClellan FIR filter design

The MATLAB command

```
b = remez(N,F,A)
```

returns the impulse response of the length `N+1` linear phase FIR filter of order `N` designed by Parks-McClellan algorithm. F is a vector of frequency band edges in ascending order between 0 and 1 with 1 corresponding to the half sampling rate. A is a real vector of the same size as F which specifies the desired amplitude of the frequency response of the points `(F(k),A(k))` and `(F(k+1),A(k+1))` for odd `k`. For odd `k`, the bands between `F(k+1)` and `F(k+2)` is considered as transition bands.

MATLAB FIR Filter Design Exercise

FIR Filter Design MATLAB Exercise

Design by windowing

Exercise:

Problem:

Assuming sampling rate at 48kHz, design an order-40 low-pass filter having cut-off frequency 10kHz by windowing method. In your design, use Hamming window as the windowing function.

Solution:

```
b = fir1(40,10.0/48.0)
```

Parks-McClellan Optimal Design

Exercise:

Problem:

Assuming sampling rate at 48kHz, design an order-40 lowpass filter having transition band 10kHz-11kHz using the Parks-McClellan optimal FIR filter design algorithm.

Solution:

```
b = remez(40,[1 1 0 0],[0 10/48 11/48 1])
```

Introduction to Random Signals and Processes

Before now, you have probably dealt strictly with the theory behind signals and systems, as well as look at some the basic characteristics of signals and systems. In doing so you have developed an important foundation; however, most electrical engineers do not get to work in this type of fantasy world. In many cases the signals of interest are very complex due to the randomness of the world around them, which leaves them noisy and often corrupted. This often causes the information contained in the signal to be hidden and distorted. For this reason, it is important to understand these random signals and how to recover the necessary information.

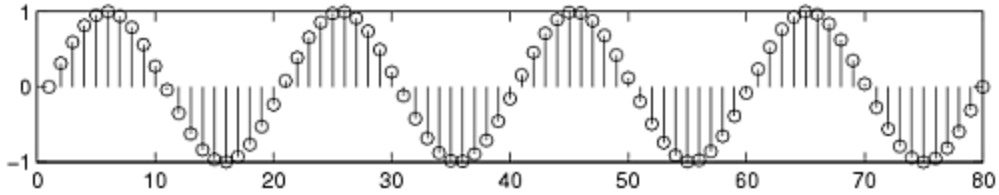
Signals: Deterministic vs. Stochastic

For this study of signals and systems, we will divide signals into two groups: those that have a fixed behavior and those that change randomly. As most of you have probably already dealt with the first type, we will focus on introducing you to random signals. Also, note that we will be dealing strictly with discrete-time signals since they are the signals we deal with in DSP and most real-world computations, but these same ideas apply to continuous-time signals.

Deterministic Signals

Most introductions to signals and systems deal strictly with **deterministic signals**. Each value of these signals are fixed and can be determined by a mathematical expression, rule, or table. Because of this, future values of any deterministic signal can be calculated from past values. For this reason, these signals are relatively easy to analyze as they do not change, and we can make accurate assumptions about their past and future behavior.

Deterministic Signal

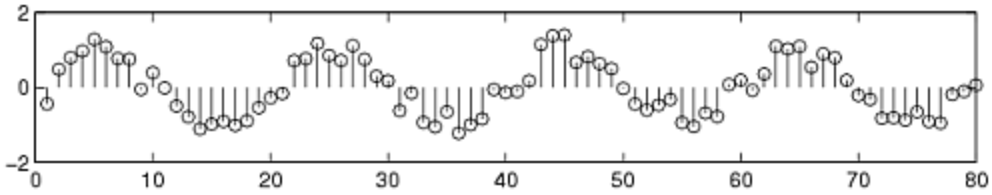


An example of a deterministic signal, the sine wave.

Stochastic Signals

Unlike deterministic signals, **stochastic signals**, or **random signals**, are not so nice. Random signals cannot be characterized by a simple, well-defined mathematical equation and their future values cannot be predicted. Rather, we must use probability and statistics to analyze their behavior. Also, because of their randomness, average values from a collection of signals are usually studied rather than analyzing one individual signal.

Random Signal



We have taken the above sine wave and added random noise to it to come up with a noisy, or random, signal. These are the types of signals that we wish to learn how to deal with so that we can recover the original sine wave.

Random Process

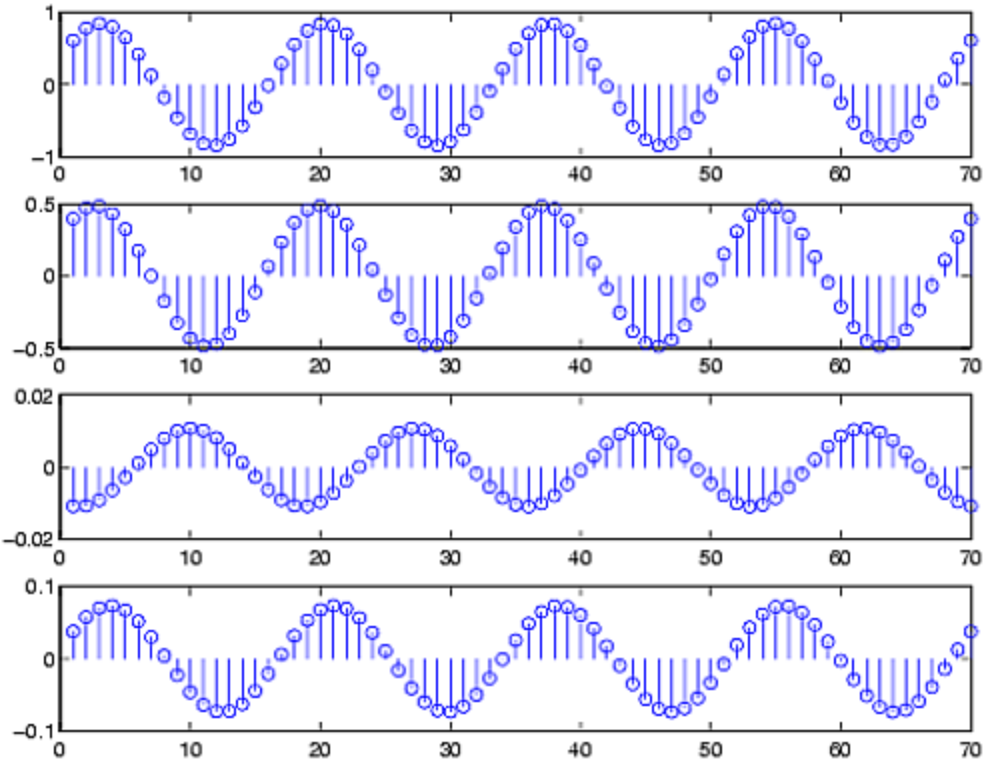
As mentioned above, in order to study random signals, we want to look at a collection of these signals rather than just one instance of that signal. This collection of signals is called a **random process**.

random process

A family or ensemble of signals that correspond to every possible outcome of a certain signal measurement. Each signal in this collection is referred to as a **realization** or **sample function** of the process.

Example:
As an example of a random process, let us look at the Random Sinusoidal Process below. We use $f[n] = A\sin(\omega n + \varphi)$ to represent the sinusoid with a given amplitude and phase. Note that the phase and amplitude of each sinusoid is based on a random number, thus making this a random process.

Random Sinusoidal Process



A random sinusoidal process, with the amplitude and phase being random numbers.

A random process is usually denoted by $X(t)$ or $X[n]$, with $x(t)$ or $x[n]$ used to represent an individual signal or waveform from this process.

In many notes and books, you might see the following notation and terms used to describe different types of random processes. For a **discrete random process**, sometimes just called a **random sequence**, t represents time that has a finite number of values. If t can take on any value of time, we have a **continuous random process**. Often times discrete and continuous refer to the amplitude of the process, and process or sequence refer to the nature of the time variable. For this study, we often just use **random process** to refer to a general collection of discrete-time signals, as seen above in [link].

Stationary and Nonstationary Random Processes

Introduction

From the definition of a random process, we know that all random processes are composed of random variables, each at its own unique point in time. Because of this, random processes have all the properties of random variables, such as mean, correlation, variances, etc.. When dealing with groups of signals or sequences it will be important for us to be able to show whether of not these statistical properties hold true for the entire random process. To do this, the concept of **stationary processes** has been developed. The general definition of a stationary process is:

stationary process
: a random process where all of its statistical properties do not vary with time

Processes whose statistical properties do change are referred to as **nonstationary**.

Understanding the basic idea of stationarity will help you to be able to follow the more concrete and mathematical definition to follow. Also, we will look at various levels of stationarity used to describe the various types of stationarity characteristics a random process can have.

Distribution and Density Functions

In order to properly define what it means to be stationary from a mathematical standpoint, one needs to be somewhat familiar with the concepts of distribution and density functions. If you can remember your statistics then feel free to skip this section!

Recall that when dealing with a single random variable, the **probability distribution function** is a simply tool used to identify the probability that our observed random variable will be less than or equal to a given number. More precisely, let X be our random variable, and let x be our given value; from this we can define the distribution function as

Equation:

$$F_x(x) = \Pr[X \leq x]$$

This same idea can be applied to instances where we have multiple random variables as well. There may be situations where we want to look at the probability of event X **and** Y both occurring. For example, below is an example of a second-order **joint distribution function**.
Equation:

$$F_x(x, y) = \Pr[X \leq x, Y \leq y]$$

While the distribution function provides us with a full view of our variable or processes probability, it is not always the most useful for calculations. Often times we will want to look at its derivative, the **probability density function (pdf)**. We define the the pdf as
Equation:

$$f_x(x) = \frac{\mathrm{d}}{\mathrm{d}\, x} F_x(x)$$

Equation:

$$f_x(x)\,\mathrm{dx} = \Pr[x < X \leq x + \mathrm{dx}]$$

[link] reveals some of the physical significance of the density function. This equations tells us the probability that our random variable falls within a given interval can be approximated by $f_x(x)$ dx. From the pdf, we can now use our knowledge of integrals to evaluate probabilities from the above approximation. Again we can also define a **joint density function** which will include multiple random variables just as was done for the distribution function. The density function is used for a variety of calculations, such as finding the expected value or proving a random variable is stationary, to name a few.

Note: The above examples explain the distribution and density functions in terms of a single random variable, X. When we are dealing with signals and random processes, remember that we will have a set of random variables where a different random variable will occur at each time instance of the random process, $X(t_k)$. In other words, the distribution and density function will also need to take into account the choice of time.

Stationarity

Below we will now look at a more in depth and mathematical definition of a stationary process. As was mentioned previously, various levels of stationarity exist and we will look at the most common types.

First-Order Stationary Process

A random process is classified as **first-order stationary** if its first-order probability density function remains equal regardless of any shift in time to its time origin. If we let x_{t_1} represent a given value at time t_1, then we define a first-order stationary as one that satisfies the following equation:
Equation:

$$f_x(x_{t_1}) = f_x(x_{t_1+\tau})$$

The physical significance of this equation is that our density function, $f_x(x_{t_1})$, is completely independent of t_1 and thus any time shift, τ.

The most important result of this statement, and the identifying characteristic of any first-order stationary process, is the fact that the mean is a constant, independent of any time shift. Below we show the results for a random process, X, that is a discrete-time signal, $x[n]$.
Equation:

$$\begin{aligned} X &= m_x[n] \\ &= E[x[n]] \\ &= \text{constant (independent of n)} \end{aligned}$$

Second-Order and Strict-Sense Stationary Process

A random process is classified as **second-order stationary** if its second-order probability density function does not vary over any time shift applied to both values. In other words, for values x_{t_1} and x_{t_2} then we will have the following be equal for an arbitrary time shift τ.
Equation:

$$f_x(x_{t_1}, x_{t_2}) = f_x(x_{t_1+\tau}, x_{t_2+\tau})$$

From this equation we see that the absolute time does not affect our functions, rather it only really depends on the time difference between the two variables. Looked at another way, this equation can be described as
Equation:

$$\Pr[X(t_1) \leq x_1, X(t_2) \leq x_2] = \Pr[X(t_1 + \tau) \leq x_1, X(t_2 + \tau) \leq x_2]$$

These random processes are often referred to as **strict sense stationary (SSS)** when **all** of the distribution functions of the process are unchanged regardless of the time shift applied to them.

For a second-order stationary process, we need to look at the autocorrelation function to see its most important property. Since we have already stated that a second-order stationary process depends only on the time difference, then all of these types of processes have the following property:
Equation:

$$\begin{aligned} R_{xx}(t, t+\tau) &= E[X(t+\tau)] \\ &= R_{xx}(\tau) \end{aligned}$$

Wide-Sense Stationary Process

As you begin to work with random processes, it will become evident that the strict requirements of a SSS process is more than is often necessary in order to adequately approximate our calculations on random processes. We define a final type of stationarity, referred to as **wide-sense stationary (WSS)**, to have slightly more relaxed requirements but ones that are still enough to provide us with adequate results. In order to be WSS a random process only needs to meet the following two requirements.

1. $X = E[x[n]] = \text{constant}$
2. $E[X(t+\tau)] = R_{xx}(\tau)$

Note that a second-order (or SSS) stationary process will always be WSS; however, the reverse will not always hold true.

Random Processes: Mean and Variance

In order to study the characteristics of a random process, let us look at some of the basic properties and operations of a random process. Below we will focus on the operations of the random signals that compose our random processes. We will denote our random process with X and a random variable from a random process or signal by x.

Mean Value

Finding the average value of a set of random signals or random variables is probably the most fundamental concepts we use in evaluating random processes through any sort of statistical method. **The mean of a random process is the average of all realizations of that process.** In order to find this average, we must look at a random signal over a range of time (possible values) and determine our average from this set of values. The **mean**, or average, of a random process, $x(t)$, is given by the following equation:
Equation:

$$\begin{aligned} m_x(t) &= \mu_x(t) \\ &= X \\ &= E[X] \\ &= \int_{-\infty}^{\infty} x f(x)\, \mathrm{d}\, x \end{aligned}$$

This equation may seem quite cluttered at first glance, but we want to introduce you to the various notations used to represent the mean of a random signal or process. Throughout texts and other readings, remember that these will all equal the same thing. The symbol, $\mu_x(t)$, and the X with a bar over it are often used as a short-hand to represent an average, so you might see it in certain textbooks. The other important notation used is, $E[X]$, which represents the "expected value of X" or the mathematical expectation. This notation is very common and will appear again.

If the random variables, which make up our random process, are discrete or quantized values, such as in a binary process, then the integrals become

summations over all the possible values of the random variable. In this case, our expected value becomes
Equation:

$$E[x[n]] = \sum_{x} \alpha \Pr[x[n] = \alpha]$$

If we have two random signals or variables, their averages can reveal how the two signals interact. If the product of the two individual averages of both signals do **not** equal the average of the product of the two signals, then the two signals are said to be **linearly independent**, also referred to as **uncorrelated**.

In the case where we have a random process in which only one sample can be viewed at a time, then we will often not have all the information available to calculate the mean using the density function as shown above. In this case we must estimate the mean through the time-average mean, discussed later. For fields such as signal processing that deal mainly with discrete signals and values, then these are the averages most commonly used.

Properties of the Mean

- The expected value of a constant, α, is the constant:
  Equation:

$$E[\alpha] = \alpha$$

- Adding a constant, α, to each term increases the expected value by that constant:
  Equation:

$$E[X + \alpha] = E[X] + \alpha$$

- Multiplying the random variable by a constant, α, multiplies the expected value by that constant.

Equation:

$$E[\alpha X] = \alpha E[X]$$

- The expected value of the sum of two or more random variables, is the sum of each individual expected value.
  Equation:

$$E[X + Y] = E[X] + E[Y]$$

Mean-Square Value

If we look at the second **moment** of the term (we now look at x^2 in the integral), then we will have the **mean-square value** of our random process. As you would expect, this is written as
Equation:

$$\begin{aligned} X^2 &= E\left[X^2\right] \\ &= \int_{-\infty}^{\infty} x^2 f(x)\ \mathrm{d}\ x \end{aligned}$$

This equation is also often referred to as the **average power** of a process or signal.

Variance

Now that we have an idea about the average value or values that a random process takes, we are often interested in seeing just how spread out the different random values might be. To do this, we look at the **variance** which is a measure of this spread. The variance, often denoted by σ^2, is written as follows:
Equation:

$$\begin{aligned} \sigma^2 &= \operatorname{Var}(X) \\ &= E\left[(X - E[X])^2\right] \\ &= \int_{-\infty}^{\infty} \left(x - X\right)^2 f(x)\, \mathrm{d}\, x \end{aligned}$$

Using the rules for the expected value, we can rewrite this formula as the following form, which is commonly seen:
Equation:

$$\begin{aligned} \sigma^2 &= X^2 - \left(X\right)^2 \\ &= E\left[X^2\right] - (E[X])^2 \end{aligned}$$

Standard Deviation

Another common statistical tool is the standard deviation. Once you know how to calculate the variance, the standard deviation is simply **the square root of the variance**, or σ.

Properties of Variance

- The variance of a constant, α, equals zero:
  Equation:

$$\begin{aligned} \operatorname{Var}(\alpha) &= \sigma(\alpha)^2 \\ &= 0 \end{aligned}$$

- Adding a constant, α, to a random variable does not affect the variance because the mean increases by the same value:
  Equation:

$$\begin{aligned}\operatorname{Var}(X+\alpha) &= \sigma(X+\alpha)^2 \\ &= \sigma(X)^2\end{aligned}$$

- Multiplying the random variable by a constant, α, increases the variance by the square of the constant:
  Equation:

$$\begin{aligned}\operatorname{Var}(\alpha X) &= \sigma(\alpha X)^2 \\ &= \alpha^2\sigma(X)^2\end{aligned}$$

- The variance of the sum of two random variables only equals the sum of the variances if the variable are **independent**.
  Equation:

$$\begin{aligned}\operatorname{Var}(X+Y) &= \sigma(X+Y)^2 \\ &= \sigma(X)^2+\sigma(Y)^2\end{aligned}$$

  Otherwise, if the random variable are **not** independent, then we must also include the covariance of the product of the variables as follows:
  Equation:

$$\operatorname{Var}(X+Y) = \sigma(X)^2 + 2\operatorname{Cov}(X,Y) + \sigma(Y)^2$$

Time Averages

In the case where we can not view the entire ensemble of the random process, we must use time averages to estimate the values of the mean and variance for the process. Generally, this will only give us acceptable results for independent and **ergodic** processes, meaning those processes in which each signal or member of the process seems to have the same statistical behavior as the entire process. The time averages will also only be taken over a finite interval since we will only be able to see a finite part of the sample.

Estimating the Mean

For the ergodic random process, $x(t)$, we will estimate the mean using the time averaging function defined as
Equation:

$$\begin{aligned} X &= E[X] \\ &= \tfrac{1}{T}\int_0^T X(t)\ \mathrm{d}\ t \end{aligned}$$

However, for most real-world situations we will be dealing with discrete values in our computations and signals. We will represent this mean as
Equation:

$$\begin{aligned} X &= E[X] \\ &= \tfrac{1}{N}\sum_{n=1}^{N} X[n] \end{aligned}$$

Estimating the Variance

Once the mean of our random process has been estimated then we can simply use those values in the following variance equation (introduced in one of the above sections)
Equation:

$${\sigma_x}^2 = X^2 - \left(X\right)^2$$

Example

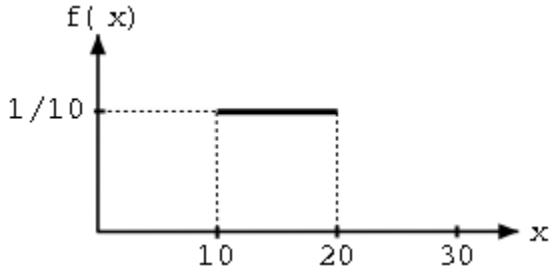
Let us now look at how some of the formulas and concepts above apply to a simple example. We will just look at a single, continuous random variable for this example, but the calculations and methods are the same for a

random process. For this example, we will consider a random variable having the probability density function described below and shown in [link].

Equation:

$$f(x) = \begin{cases} \frac{1}{10} & \text{if } 10 \leq x \leq 20 \\ 0 & \text{otherwise} \end{cases}$$

Probability Density Function



A uniform probability density function.

First, we will use [link] to solve for the mean value.

Equation:

$$\begin{aligned} X &= \int_{10}^{20} x \frac{1}{10} \, \mathrm{d}\, x \\ &= \frac{1}{10} \frac{x^2}{2} \Big|_{x=10}^{20} \\ &= \frac{1}{10} (200 - 50) \\ &= 15 \end{aligned}$$

Using [link] we can obtain the mean-square value for the above density function.

Equation:

$$
\begin{aligned}
X^2 &= \int_{10}^{20} x^2 \tfrac{1}{10} \, \mathrm{d}\, x \\
&= \tfrac{1}{10} \tfrac{x^3}{3} \Big|_{x=10}^{20} \\
&= \tfrac{1}{10} \left(\tfrac{8000}{3} - \tfrac{1000}{3} \right) \\
&= 233.33
\end{aligned}
$$

And finally, let us solve for the variance of this function.
Equation:

$$
\begin{aligned}
\sigma^2 &= X^2 - \left(X \right)^2 \\
&= 233.33 - 15^2 \\
&= 8.33
\end{aligned}
$$

Correlation and Covariance of a Random Signal

When we take the expected value, or average, of a random process, we measure several important characteristics about how the process behaves in general. This proves to be a very important observation. However, suppose we have several random processes measuring different aspects of a system. The relationship between these different processes will also be an important observation. The covariance and correlation are two important tools in finding these relationships. Below we will go into more details as to what these words mean and how these tools are helpful. Note that much of the following discussions refer to just random variables, but keep in mind that these variables can represent random signals or random processes.

Covariance

To begin with, when dealing with more than one random process, it should be obvious that it would be nice to be able to have a number that could quickly give us an idea of how similar the processes are. To do this, we use the **covariance**, which is analogous to the variance of a single variable.

Covariance
: A measure of how much the deviations of two or more variables or processes match.

For two processes, X and Y, if they are **not** closely related then the covariance will be small, and if they are similar then the covariance will be large. Let us clarify this statement by describing what we mean by "related" and "similar." Two processes are "closely related" if their distribution spreads are almost equal and they are around the same, or a very slightly different, mean.

Mathematically, covariance is often written as σ_{xy} and is defined as
Equation:

$$\begin{aligned} \operatorname{cov}(X,Y) &= \sigma_{xy} \\ &= E\Big[\Big(X-X\Big)\Big(Y-Y\Big)\Big] \end{aligned}$$

This can also be reduced and rewritten in the following two forms:
Equation:

$$\sigma_{xy} = (xy) - \bar{x}\bar{y}$$

Equation:

$$\sigma_{xy} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left(X - X\right) \left(Y - Y\right) f(x, y) \, \mathrm{d}\, x \, \mathrm{d}\, y$$

Useful Properties

- If X and Y are independent and uncorrelated or one of them has zero mean value, then

$$\sigma_{xy} = 0$$

- If X and Y are orthogonal, then

$$\sigma_{xy} = -\left(E[X]E[Y]\right)$$

- The covariance is symmetric

$$\operatorname{cov}\left(X, Y\right) = \operatorname{cov}\left(Y, X\right)$$

- Basic covariance identity

$$\operatorname{cov}\left(X + Y, Z\right) = \operatorname{cov}\left(X, Z\right) + \operatorname{cov}\left(Y, Z\right)$$

- Covariance of equal variables

$$\operatorname{cov}\left(X, X\right) = \operatorname{Var}\left(X\right)$$

Correlation

For anyone who has any kind of statistical background, you should be able to see that the idea of dependence/independence among variables and signals plays an important role when dealing with random processes. Because of this, the **correlation** of two variables provides us with a measure of how the two variables affect one another.

Correlation

A measure of how much one random variable depends upon the other.

This measure of association between the variables will provide us with a clue as to how well the value of one variable can be predicted from the value of the other. The correlation is equal to the average of the product of two random variables and is defined as

Equation:

$$\begin{aligned} \operatorname{cor}(X,Y) &= E[XY] \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyf(x,y)\,\mathrm{d}\,x\,\mathrm{d}\,y \end{aligned}$$

Correlation Coefficient

It is often useful to express the correlation of random variables with a range of numbers, like a percentage. For a given set of variables, we use the **correlation coefficient** to give us the linear relationship between our variables. The correlation coefficient of two variables is defined in terms of their covariance and standard deviations, denoted by σ_x, as seen below

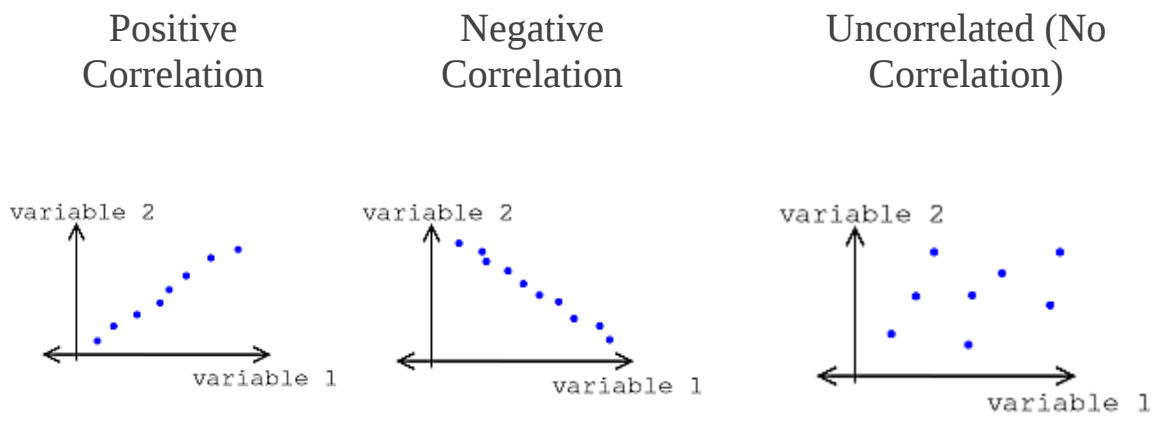
Equation:

$$\rho = \frac{\operatorname{cov}(X,Y)}{\sigma_x \sigma_y}$$

where we will always have

$$-1 \leq \rho \leq 1$$

This provides us with a quick and easy way to view the correlation between our variables. If there is no relationship between the variables then the correlation coefficient will be zero and if there is a perfect positive match it will be one. If there is a perfect inverse relationship, where one set of variables increases while the other decreases, then the correlation coefficient will be negative one. This type of correlation is often referred to more specifically as the **Pearson's Correlation Coefficient**,or Pearson's Product Moment Correlation.



Types of Correlation

Note: So far we have dealt with correlation simply as a number relating the relationship between any two variables. However, since our goal will be to relate random processes to each other, which deals with signals as a function of time, we will want to continue this study by looking at **correlation functions**.

Example

Now let us take just a second to look at a simple example that involves calculating the covariance and correlation of two sets of random numbers. We are given the following data sets:

$$x = \{3, 1, 6, 3, 4\}$$

$$y = \{1, 5, 3, 4, 3\}$$

To begin with, for the covariance we will need to find the expected value, or mean, of x and y.

$$\bar{x} = \frac{1}{5}\left(3 + 1 + 6 + 3 + 4\right) = 3.4$$

$$\bar{y} = \frac{1}{5}\left(1 + 5 + 3 + 4 + 3\right) = 3.2$$

$$xy = \frac{1}{5}\left(3 + 5 + 18 + 12 + 12\right) = 10$$

Next we will solve for the standard deviations of our two sets using the formula below (for a review click here).

$$\sigma = \sqrt{E\left[\left(X - E[X]\right)^2\right]}$$

$$\sigma_x = \sqrt{\frac{1}{5}\left(0.16 + 5.76 + 6.76 + 0.16 + 0.36\right)} = 1.625$$

$$\sigma_y = \sqrt{\frac{1}{6}\left(4.84 + 3.24 + 0.04 + 0.64 + 0.04\right)} = 1.327$$

Now we can finally calculate the covariance using one of the two formulas found above. Since we calculated the three means, we will use that formula since it will be much simpler.

$$\sigma_{xy} = 10 - 3.4 \times 3.2 = -0.88$$

And for our last calculation, we will solve for the correlation coefficient, ρ.

$$\rho = \frac{-0.88}{1.625 \times 1.327} = -0.408$$

Matlab Code for Example

The above example can be easily calculated using Matlab. Below I have included the code to find all of the values above.

```
x = [3 1 6 3 4];
y = [1 5 3 4 3];

mx = mean(x)
my = mean(y)
mxy = mean(x.*y)

% Standard Dev. from built-in Matlab
Functions
std(x,1)
std(y,1)

% Standard Dev. from Equation Above
(same result as std(?,1))
sqrt( 1/5 * sum((x-mx).^2))
sqrt( 1/5 * sum((y-my).^2))

cov(x,y,1)

corrcoef(x,y)
```

Autocorrelation of Random Processes

Before diving into a more complex statistical analysis of random signals and processes, let us quickly review the idea of correlation. Recall that the correlation of two signals or variables is the expected value of the product of those two variables. Since our focus will be to discover more about a random process, a collection of random signals, then imagine us dealing with two samples of a random process, where each sample is taken at a different point in time. Also recall that the key property of these random processes is that they are now functions of time; imagine them as a collection of signals. The expected value of the product of these two variables (or samples) will now depend on how quickly they change in regards to **time**. For example, if the two variables are taken from almost the same time period, then we should expect them to have a high correlation. We will now look at a correlation function that relates a pair of random variables from the same process to the time separations between them, where the argument to this correlation function will be the time difference. For the correlation of signals from two different random process, look at the crosscorrelation function.

Autocorrelation Function

The first of these correlation functions we will discuss is the **autocorrelation**, where each of the random variables we will deal with come from the same random process.

Autocorrelation
: the expected value of the product of a random variable or signal realization with a time-shifted version of itself

With a simple calculation and analysis of the autocorrelation function, we can discover a few important characteristics about our random process. These include:

1. How quickly our random signal or processes changes with respect to the time function

2. Whether our process has a periodic component and what the expected frequency might be

As was mentioned above, the autocorrelation function is simply the expected value of a product. Assume we have a pair of random variables from the same process, $X_1 = X(t_1)$ and $X_2 = X(t_2)$, then the autocorrelation is often written as
Equation:

$$\begin{aligned} R_{\text{xx}}(t_1, t_2) &= E[X_1 X_2] \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x_1 x_2 f(x_1, x_2) \, \mathrm{d}\, x_2 \, \mathrm{d}\, x_1 \end{aligned}$$

The above equation is valid for stationary and nonstationary random processes. For stationary processes, we can generalize this expression a little further. Given a wide-sense stationary processes, it can be proven that the expected values from our random process will be independent of the origin of our time function. Therefore, we can say that our autocorrelation function will depend on the time difference and not some absolute time. For this discussion, we will let $\tau = t_2 - t_1$, and thus we generalize our autocorrelation expression as
Equation:

$$\begin{aligned} R_{\text{xx}}(t, t+\tau) &= R_{\text{xx}}(\tau) \\ &= E[X(t) X(t+\tau)] \end{aligned}$$

for the continuous-time case. In most DSP course we will be more interested in dealing with real signal sequences, and thus we will want to look at the discrete-time case of the autocorrelation function. The formula below will prove to be more common and useful than [link]:
Equation:

$$R_{\text{xx}}[n, n+m] = \sum_{n=-\infty}^{\infty} x[n] x[n+m]$$

And again we can generalize the notation for our autocorrelation function as **Equation:**

$$\begin{aligned} R_{\text{xx}}[n, n+m] &= R_{\text{xx}}[m] \\ &= E[X[n]X[n+m]] \end{aligned}$$

Properties of Autocorrelation

Below we will look at several properties of the autocorrelation function that hold for **stationary** random processes.

- Autocorrelation is an even function for τ

$$R_{\text{xx}}(\tau) = R_{\text{xx}}(-\tau)$$

- The mean-square value can be found by evaluating the autocorrelation where $\tau = 0$, which gives us

$$R_{\text{xx}}(0) = X^2$$

- The autocorrelation function will have its largest value when $\tau = 0$. This value can appear again, for example in a periodic function at the values of the equivalent periodic points, but will never be exceeded.

$$R_{\text{xx}}(0) \geq |R_{\text{xx}}(\tau)|$$

- If we take the autocorrelation of a period function, then $R_{\text{xx}}(\tau)$ will also be periodic with the same frequency.

Estimating the Autocorrleation with Time-Averaging

Sometimes the whole random process is not available to us. In these cases, we would still like to be able to find out some of the characteristics of the

stationary random process, even if we just have part of one sample function. In order to do this we can **estimate** the autocorrelation from a given interval, 0 to T seconds, of the sample function.
Equation:

$$\check{R}_{\mathrm{xx}}(\tau) = \frac{1}{T-\tau} \int_0^{T-\tau} x(t)x(t+\tau)\ \mathrm{d}\,t$$

However, a lot of times we will not have sufficient information to build a complete continuous-time function of one of our random signals for the above analysis. If this is the case, we can treat the information we do know about the function as a discrete signal and use the discrete-time formula for estimating the autocorrelation.
Equation:

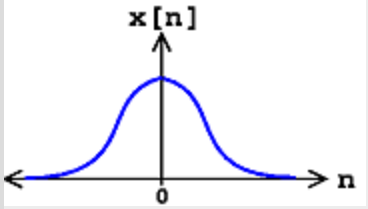
$$\check{R}_{\mathrm{xx}}[m] = \frac{1}{N-m} \sum_{n=0}^{N-m-1} x[n]x[n+m]$$

Examples

Below we will look at a variety of examples that use the autocorrelation function. We will begin with a simple example dealing with Gaussian White Noise (GWN) and a few basic statistical properties that will prove very useful in these and future calculations.

Example:
We will let $x[n]$ represent our GWN. For this problem, it is important to remember the following fact about the mean of a GWN function:

$$E[x[n]] = 0$$



Gaussian density function. By examination, can easily see that the above statement is true - the mean equals zero.

Along with being **zero-mean**, recall that GWN is always **independent**. With these two facts, we are now ready to do the short calculations required to find the autocorrelation.

$$R_{\text{xx}}[n, n+m] = E[x[n]x[n+m]]$$

Since the function, $x[n]$, is independent, then we can take the product of the individual expected values of both functions.

$$R_{\text{xx}}[n, n+m] = E[x[n]]E[x[n+m]]$$

Now, looking at the above equation we see that we can break it up further into two conditions: one when m and n are equal and one when they are not equal. When they are equal we can combine the expected values. We are left with the following piecewise function to solve:

$$R_{\text{xx}}[n, n+m] = \begin{cases} E[x[n]]E[x[n+m]] & \text{if } \ m \neq 0 \\ E\left[x^2[n]\right] & \text{if } \ m = 0 \end{cases}$$

We can now solve the two parts of the above equation. The first equation is easy to solve as we have already stated that the expected value of $x[n]$ will be zero. For the second part, you should recall from statistics that the

expected value of the square of a function is equal to the variance. Thus we get the following results for the autocorrelation:

$$R_{\text{xx}}[n, n+m] = \begin{cases} 0 \ \text{ if } \ m \neq 0 \\ \sigma^2 \ \text{ if } \ m = 0 \end{cases}$$

Or in a more concise way, we can represent the results as

$$R_{\text{xx}}[n, n+m] = \sigma^2 \delta[m]$$

Crosscorrelation of Random Processes

Before diving into a more complex statistical analysis of random signals and processes, let us quickly review the idea of correlation. Recall that the correlation of two signals or variables is the expected value of the product of those two variables. Since our main focus is to discover more about random processes, a collection of random signals, we will deal with two random processes in this discussion, where in this case we will deal with samples from two **different** random processes. We will analyze the expected value of the product of these two variables and how they correlate to one another, where the argument to this correlation function will be the time difference. For the correlation of signals from the same random process, look at the autocorrelation function.

Crosscorrelation Function

When dealing with multiple random processes, it is also important to be able to describe the relationship, if any, between the processes. For example, this may occur if more than one random signal is applied to a system. In order to do this, we use the **crosscorrelation function**, where the variables are instances from two different wide sense stationary random processes.

Crosscorrelation
: if two processes are wide sense stationary, the expected value of the product of a random variable from one random process with a time-shifted, random variable from a different random process

Looking at the generalized formula for the crosscorrelation, we will represent our two random processes by allowing $U = U(t)$ and $V = V(t - \tau)$. We will define the crosscorrelation function as
Equation:

$$\begin{aligned} R_{uv}(t, t-\tau) &= E[UV] \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} uvf(u,v) \, \mathrm{d}\, v \, \mathrm{d}\, u \end{aligned}$$

Just as the case with the autocorrelation function, if our input and output, denoted as $U(t)$ and $V(t)$, are at least jointly wide sense stationary, then the crosscorrelation does not depend on absolute time; it is just a function of the time difference. This means we can simplify our writing of the above function as

Equation:

$$R_{uv}(\tau) = E[UV]$$

or if we deal with two real signal sequences, $x[n]$ and $y[n]$, then we arrive at a more commonly seen formula for the discrete crosscorrelation function. See the formula below and notice the similarities between it and the convolution of two signals:

Equation:

$$\begin{aligned} R_{xy}(n, n-m) &= R_{xy}(m) \\ &= \sum_{n=-\infty}^{\infty} x[n]y[n-m] \end{aligned}$$

Properties of Crosscorrelation

Below we will look at several properties of the crosscorrelation function that hold for two **wide sense stationary (WSS)** random processes.

- Crosscorrelation is **not** an even function; however, it does have a unique symmetry property:
  Equation:

  $$R_{xy}(-\tau) = R_{yx}(\tau)$$

- The maximum value of the crosscorrelation is not always when the shift equals zero; however, we can prove the following property revealing to us what value the maximum cannot exceed.
  Equation:

$$|R_{xy}(\tau)| \leq \sqrt{R_{xx}(0)R_{yy}(0)}$$

- When two random processes are statistically independent then we have **Equation:**

$$R_{xy}(\tau) = R_{yx}(\tau)$$

Examples

Exercise:

Problem:

Let us begin by looking at a simple example showing the relationship between two sequences. Using [link], find the crosscorrelation of the sequences

$$x[n] = \{\ldots, 0, 0, 2, -3, 6, 1, 3, 0, 0, \ldots\}$$

$$y[n] = \{\ldots, 0, 0, 1, -2, 4, 1, -3, 0, 0, \ldots\}$$

for each of the following possible time shifts: $m = \{0, 3, -1\}$.

Solution:

1. For $m = 0$, we should begin by finding the product sequence $s[n] = x[n]y[n]$. Doing this we get the following sequence:

   $$s[n] = \{\ldots, 0, 0, 2, 6, 24, 1, -9, 0, 0, \ldots\}$$

   and so from the sum in our crosscorrelation function we arrive at the answer of

   $$R_{xy}(0) = 22$$

2. For $m = 3$, we will approach it the same was we did above; however, we will now shift $y[n]$ to the right. Then we can find the

product sequence $s[n] = x[n]y[n-3]$, which yields

$$s[n] = \{\ldots, 0, 0, 0, 0, 0, 1, -6, 0, 0, \ldots\}$$

and from the crosscorrelation function we arrive at the answer of

$$R_{xy}(3) = -6$$

3. For $m = -1$, we will again take the same approach; however, we will now shift $y[n]$ to the left. Then we can find the product sequence $s[n] = x[n]y[n+1]$, which yields

$$s[n] = \{\ldots, 0, 0, -4, -12, 6, -3, 0, 0, 0, \ldots\}$$

and from the crosscorrelation function we arrive at the answer of

$$R_{xy}(-1) = -13$$

Introduction to Adaptive Filters

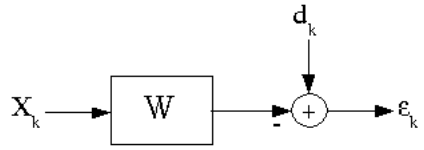
In many applications requiring filtering, the necessary frequency response may not be known beforehand, or it may vary with time. (Example; suppression of engine harmonics in a car stereo.) In such applications, an adaptive filter which can automatically design itself and which can track system variations in time is extremely useful. Adaptive filters are used extensively in a wide variety of applications, particularly in telecommunications.

Outline of adaptive filter material

1. **Wiener Filters** L optimal (FIR) filter design in a statistical context
2. **LMS algorithm** simplest and by-far-the-most-commonly-used adaptive filter algorithm
3. **Stability and performance of the LMS algorithm** When and how well it works
4. **Applications of adaptive filters** Overview of important applications
5. **Introduction to advanced adaptive filter algorithms** Techniques for special situations or faster convergence

Discrete-Time, Causal Wiener Filter

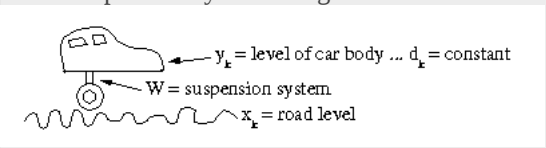
Stochastic L_2 optimal (least squares) FIR filter design problem: Given a wide-sense stationary (WSS) input signal x_k and desired signal d_k (WSS $\Leftrightarrow E[y_k] = E[y_{k+d}]$, $r_{yz}(l) = E[y_k z_{k+l}]$, $\forall k, l : (r_{yy}(0) < \infty)$)



The Wiener filter is the linear, time-invariant filter minimizing $E\left[\varepsilon^2\right]$, the variance of the error.

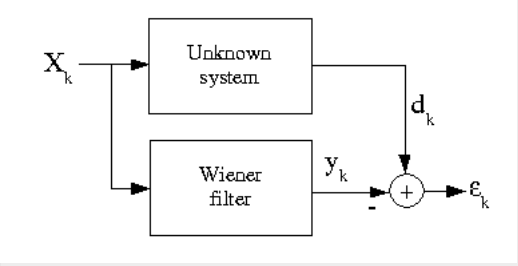
As posed, this problem seems slightly silly, since d_k is already available! However, this idea is useful in a wide cariety of applications.

Example:
active suspension system design



Note:optimal system may change with different road conditions or mass in car, so an **adaptive** system might be desirable.

Example:
System identification (radar, non-destructive testing, adaptive control systems)



Exercise:

Problem:

Usually one desires that the input signal x_k be "persistently exciting," which, among other things, implies non-zero energy in all frequency bands. Why is this desirable?

Determining the optimal length-N causal FIR Weiner filter

Note: for convenience, we will analyze only the causal, real-data case; extensions are straightforward.

$$y_k = \sum_{l=0}^{M-1} w_l x_{k-l}$$

$$\underset{w_l}{\operatorname{argmin}}\, E\left[\varepsilon^2\right] = E\left[(d_k - y_k)^2\right] = E\left[\left(d_k - \sum_{l=0}^{M-1} w_l x_{k-l}\right)^2\right] = E\left[{d_k}^2\right] - 2\sum_{l=0}^{M-1} w_l E[d_k x_{k-l}] + \sum_{l=0}^{M-1}\sum_{m=0}^{M-1}(w_l$$

$$E\left[\varepsilon^2\right] = r_{\mathrm{dd}}(0) - 2\sum_{l=0}^{M-1} w_l r_{\mathrm{dx}}(l) + \sum_{l=0}^{M-1}\sum_{m=0}^{M-1} w_l w_m r_{\mathrm{xx}}(l-m)$$

where

$$r_{\mathrm{dd}}(0) = E\left[{d_k}^2\right]$$

$$r_{\mathrm{dx}}(l) = E[d_k X_{k-l}]$$

$$r_{\mathrm{xx}}(l-m) = E[x_k x_{k+l-m}]$$

This can be written in matrix form as

$$E\left[\varepsilon^2\right] = r_{\mathrm{dd}}(0) - 2\boldsymbol{P}\boldsymbol{W}^T + \boldsymbol{W}^T R \boldsymbol{W}$$

where

$$\boldsymbol{P} = \begin{pmatrix} r_{\mathrm{dx}}(0) \\ r_{\mathrm{dx}}(1) \\ \vdots \\ r_{\mathrm{dx}}(M-1) \end{pmatrix}$$

$$R = \begin{pmatrix} r_{\mathrm{xx}}(0) & r_{\mathrm{xx}}(1) & \dots & \dots & r_{\mathrm{xx}}(M-1) \\ r_{\mathrm{xx}}(1) & r_{\mathrm{xx}}(0) & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & r_{\mathrm{xx}}(0) & r_{\mathrm{xx}}(1) \\ r_{\mathrm{xx}}(M-1) & \dots & \dots & r_{\mathrm{xx}}(1) & r_{\mathrm{xx}}(0) \end{pmatrix}$$

To solve for the optimum filter, compute the gradient with respect to the top weights vector W

$$\nabla \doteq \begin{pmatrix} \frac{\partial \varepsilon^2}{\partial w_0} \\ \frac{\partial \varepsilon^2}{\partial w_1} \\ \vdots \\ \frac{\partial \varepsilon^2}{\partial w_{M-1}} \end{pmatrix}$$

$$\nabla = -\left(2\boldsymbol{P}\right) + 2R\boldsymbol{W}$$

(recall $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}}\left(\boldsymbol{A}^T W\right) = \boldsymbol{A}^T$, $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}}(\boldsymbol{W}M\boldsymbol{W}) = 2M\boldsymbol{W}$ for symmetric M) setting the gradient equal to zero ⇒

$$W_{\mathrm{opt}}R = \boldsymbol{P} \Rightarrow W_{\mathrm{opt}} = R^{-1}\boldsymbol{P}$$

Since R is a correlation matrix, it must be non-negative definite, so this is a minimizer. For R positive definite, the minimizer is unique.

Practical Issues in Wiener Filter Implementation

The weiner-filter, $W_{\text{opt}} = R^{-1}\boldsymbol{P}$, is ideal for many applications. But several issues must be addressed to use it in practice.

Exercise:

Problem:

In practice one usually won't know exactly the statistics of x_k and d_k (i.e. R and $\boldsymbol{P}$) needed to compute the Weiner filter.

How do we surmount this problem?

Solution:

Estimate the statistics

$$\widehat{r_{\text{xx}}(l)} \simeq \frac{1}{N}\sum_{k=0}^{N-1} x_k x_{k+l}$$

$$\widehat{r_{\text{xd}}(l)} \simeq \frac{1}{N}\sum_{k=0}^{N-1} d_k x_{k-l}$$

then solve $\widehat{W}_{\text{opt}} = \widehat{R^{-1}} = \widehat{P}$

Exercise:

Problem:

In many applications, the statistics of x_k, d_k vary slowly with time.

How does one develop an **adaptive** system which tracks these changes over time to keep the system near optimal at all times?

Solution:

Use short-time windowed estiamtes of the correlation functions.

Note:

$$\left(\widehat{r_{\mathrm{xx}}(l)}\right)^k = \frac{1}{N}\sum_{m=0}^{N-1} x_{k-m}x_{k-m-l}$$

$$\left(\widehat{r_{\mathrm{dx}}(l)}\right)^k = \frac{1}{N}\sum_{m=0}^{N-1} x_{k-m-l}d_{k-m}$$

and ${W_{\mathrm{opt}}}^k \simeq \left(\widehat{R}_k\right)^{-1}\widehat{P}_k$

Exercise:

Problem: How can $\widehat{r_{\mathrm{xx}}^k(l)}$ be computed efficiently?

Solution:

Recursively!

$$r_{\mathrm{xx}}^k(l) = r_{\mathrm{xx}}^{k-1}(l) + x_k x_{k-l} - x_{k-N}x_{k-N-l}$$

This is critically stable, so people usually do

$$(1-\alpha)\left({r_{\mathrm{xx}}}^k(l) = \alpha r_{\mathrm{xx}}^{k-1}(l) + x_k x_{k-l}\right)$$

Exercise:

Problem: how does one choose N?

Tradeoffs

Larger $N \rightarrow$ more accurate estimates of the correlation values $\rightarrow$ better $\widehat{W}_{\text{opt}}$. However, larger N leads to slower adaptation.

Note: The success of adaptive systems depends on x, d being roughly stationary over at least N samples, $N > M$. That is, all adaptive filtering algorithms require that the underlying system varies slowly with respect to the sampling rate and the filter length (although they can tolerate occasional step discontinuities in the underlying system).

Computational Considerations

As presented here, an adaptive filter requires computing a matrix inverse at each sample. Actually, since the matrix R is Toeplitz, the linear system of equations can be sovled with $O\left(M^2\right)$ computations using Levinson's algorithm, where M is the filter length. However, in many applications this may be too expensive, especially since computing the filter output itself requires $O(M)$ computations. There are two main approaches to resolving the computation problem

1. Take advantage of the fact that R^{k+1} is only slightly changed from R^k to reduce the computation to $O(M)$; these algorithms are called Fast Recursive Least Squareds algorithms; all methods proposed so far have stability problems and are dangerous to use.
2. Find a different approach to solving the optimization problem that doesn't require explicit inversion of the correlation matrix.

Note: Adaptive algorithms involving the correlation matrix are called **Recursive least Squares** (RLS) algorithms. Historically, they were developed after the LMS algorithm, which is the slimplest and most widely used approach $O(M)$. $O\left(M^2\right)$ RLS algorithms are used in applications requiring very fast adaptation.

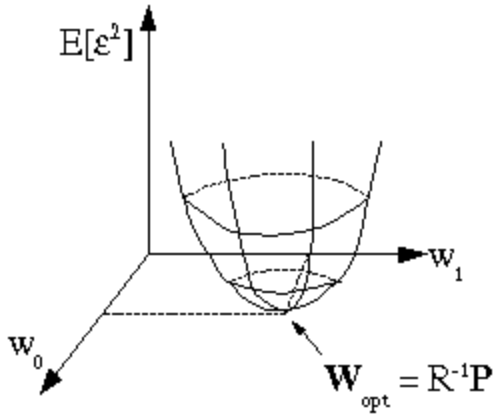
Quadratic Minimization and Gradient Descent

Quadratic minimization problems

The least squares optimal filter design problem is quadratic in the filter coefficients:

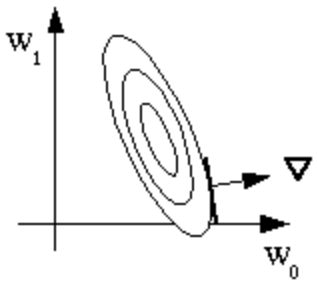
$$E\left[\varepsilon^2\right] = r_{\mathrm{dd}}(0) - 2\boldsymbol{P}^T\boldsymbol{W} + \boldsymbol{W}^T R\boldsymbol{W}$$

If R is positive definite, the error surface $E\left[\varepsilon^2\right](w_0, w_1, \ldots, w_{M-1})$ is a unimodal "bowl" in $\mathbb{R}^N$.



The problem is to find the bottom of the bowl. In an adaptive filter context, the shape and bottom of the bowl may drift slowly with time; hopefully slow enough that the adaptive algorithm can track it.

For a quadratic error surface, the bottom of the bowl can be found in one step by computing $R^{-1}\boldsymbol{P}$. Most modern nonlinear optimization methods (which are used, for example, to solve the L^P optimal IIR filter design problem!) locally approximate a nonlinear function with a second-order (quadratic) Taylor series approximation and step to the bottom of this quadratic approximation on each iteration. However, an older and simpler appraoch to nonlinear optimaztion exists, based on **gradient descent**.
Contour plot of ε-squared



The idea is to iteratively find the minimizer by computing the gradient of the error function: $E\nabla = \frac{\partial E[\varepsilon^2]}{\partial w_i}$. The gradient is a vector in $\mathbb{R}^M$ pointing in the steepest uphill direction on the error surface at a given point $\boldsymbol{W}^i$, with ∇ having a magnitude proportional to the slope of the error surface in this steepest direction.

By updating the coefficient vector by taking a step **opposite** the gradient direction : $\boldsymbol{W}^{i+1} = \boldsymbol{W}^i - \mu\nabla^i$, we go (locally) "downhill" in the steepest direction, which seems to be a sensible way to iteratively solve a nonlinear optimization problem. The performance obviously depends on μ; if μ is too large, the iterations could bounce back and forth up out of the bowl. However, if μ is too small, it could take many iterations to approach the bottom. We will determine criteria for choosing μ later.

In summary, the gradient descent algorithm for solving the Weiner filter problem is:

$$\text{Guess } W^0$$

$$\text{do } i = 1,\infty$$

$$\nabla^i = -(2P) + 2RW^i$$

$$W^{i+1} = W^i - \mu\nabla^i$$

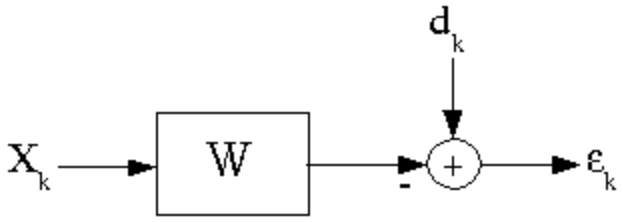
$$\text{repeat}$$

$$W_{\text{opt}} = W^\infty$$

The gradient descent idea is used in the LMS adaptive fitler algorithm. As presented, this alogrithm costs $O\left(M^2\right)$ computations per iteration and doesn't appear very attractive, but LMS only requires $O(M)$ computations and is stable, so it is very attractive when computation is an issue, even thought it converges more slowly then the RLS algorithms we have discussed so far.

The LMS Adaptive Filter Algorithm

Recall the Weiner filter problem



$\{x_k\}$, $\{d_k\}$ jointly wide sense stationary

Find W minimizing $E\left[{\varepsilon_k}^2\right]$

$$\varepsilon_k = d_k - y_k = d_k - \sum_{i=0}^{M-1} w_i x_{k-i} = d_k - {X^k}^T W^k$$

$$X^k = \begin{matrix} x_k \\ x_{k-1} \\ \vdots \\ x_{k-M+1} \end{matrix}$$

$$W^k = \begin{matrix} w_0^k \\ w_1^k \\ \vdots \\ w_{M-1}^k \end{matrix}$$

The superscript denotes absolute time, and the subscript denotes time or a vector index.

the solution can be found by setting the gradient 0
Equation:

$$
\begin{aligned}
\nabla^k &= \frac{\partial E\left[{\varepsilon_k}^2\right]}{\partial W} \\
&= E\left[2\varepsilon_k\left(-X^k\right)\right] \\
&= E\left[-2\left(d_k - {X^k}^T W_k\right)X^k\right] \\
&= -\left(2E\left[d_k X^k\right]\right) + E\left[{X^k}^T\right]W \\
&= 2P + 2RW
\end{aligned}
$$

$$
\Rightarrow \left(W_{\text{opt}} = R^{-1}P\right)
$$

Alternatively, W_{opt} can be found iteratively using a gradient descent technique

$$
W^{k+1} = W^k - \mu\nabla^k
$$

In practice, we don't know R and P exactly, and in an adaptive context they may be slowly varying with time.

To find the (approximate) Wiener filter, some approximations are necessary. As always, the key is to make the **right** approximations!

Note: Approximate R and P: ⇒ RLS methods, as discussed last time.

Note: Approximate the gradient!

$$
\nabla^k = \frac{\partial E\left[{\varepsilon_k}^2\right]}{\partial W}
$$

Note that ${\varepsilon_k}^2$ itself is a very noisy approximation to $E\left[{\varepsilon_k}^2\right]$. We can get a noisy approximation to the gradient by finding the gradient of ${\varepsilon_k}^2$! Widrow and Hoff first published the LMS algorithm, based on this clever idea, in 1960.

$$\widehat{\nabla^k} = \frac{\partial {\varepsilon_k}^2}{\partial W} = 2\varepsilon_k \frac{\partial \left(d_k - {W^k}^T X^k\right)}{\partial W} = 2\varepsilon_k \left(-X^k\right) = -\left(2\varepsilon_k X^k\right)$$

This yields the LMS adaptive filter algorithm

Example:
The LMS Adaptive Filter Algorithm

1. $y_k = {W^k}^T X^k = \sum_{i=0}^{M-1} w_i^k x_{k-i}$
2. $\varepsilon_k = d_k - y_k$
3. $W^{k+1} = W^k - \mu\widehat{\nabla^k} = W^k - \mu\left(-2\varepsilon_k X^k\right) = W^k + 2\mu\varepsilon_k X^k$ ($w_i^{k+1} = w_i^k + 2\mu\varepsilon_k x_{k-i}$)

The LMS algorithm is often called a **stochastic gradient** algorithm, since $\widehat{\nabla^k}$ is a noisy gradient. This is **by far** the most commonly used adaptive filtering algorithm, because

1. it was the first
2. it is very simple
3. in practice it works well (except that sometimes it converges slowly)
4. it requires relatively litle computation
5. it updates the tap weights every sample, so it continually adapts the filter
6. it tracks slow changes in the signal statistics well

Computational Cost of LMS

To Compute ⇒	y_k	ε_k	W^{k+1}	= **Total**
multiplies	M	0	$M+1$	$2M+1$
adds	$M-1$	1	M	$2M$

So the LMS algorithm is $O(M)$ per sample. In fact, it is nicely balanced in that the filter computation and the adaptation require the same amount of computation.

Note that the parameter μ plays a very important role in the LMS algorithm. It can also be varied with time, but usually a constant μ ("convergence weight facor") is used, chosen after experimentation for a given application.

Tradeoffs

large μ: fast convergence, fast adaptivity

small μ: accurate $W \rightarrow$ less misadjustment error, stability

First Order Convergence Analysis of the LMS Algorithm

Analysis of the LMS algorithm

It is important to analyze the LMS algorithm to determine under what conditions it is stable, whether or not it converges to the Wiener solution, to determine how quickly it converges, how much degredation is suffered due to the noisy gradient, etc. In particular, we need to know how to choose the parameter μ.

Mean of W

does W^k, $k \to \infty$ approach the Wiener solution? (since W^k is always somewhat random in the approximate gradient-based LMS algorithm, we ask whether the expected value of the filter coefficients converge to the Wiener solution)
Equation:

$$\begin{aligned} E\left[W^{k+1}\right] &= \overline{W^{k+1}} \\ &= E\left[W^k + 2\mu\varepsilon_k X^k\right] \\ &= \overline{W^k} + 2\mu E\left[d_k X^k\right] + 2\mu E\left[-\left(\left(W^{k^T} X^k\right) X^k\right)\right] \\ &= \overline{W^k} + 2\mu P + -\left(2\mu E\left[\left(W^{k^T} X^k\right) X^k\right]\right) \end{aligned}$$

Patently False Assumption

X^k and X^{k-i}, X^k and d^{k-i}, and d_k and d_{k-i} are statistically independent, $i \neq 0$. This assumption is obviously false, since X^{k-1} is the same as X^k except for shifting down the vector elements one place and adding one new sample. We make this assumption because otherwise it becomes extremely difficult to analyze the LMS algorithm. (First good analysis not making this assumption: Macchi and Eweda) Many simulations and much practical experience has shown that the results one obtains with analyses based on the patently false assumption above are quite accurate in most situations

With the independence assumption, W^k (which depends only on previous X^{k-i}, d^{k-i}) is statitically independent of X^k, and we can simplify $E\Big[\Big({W^k}^T X^k\Big)X^k\Big]$

Now $\Big({W^k}^T X^k\Big)X^k$ is a vector, and
Equation:

$$\begin{aligned} E\Big[\Big({W^k}^T X^k\Big)X^k\Big] &= E\left[\begin{pmatrix} \vdots \\ \sum_{i=0}^{M-1} w_i^k x_{k-i} x_{k-j} \\ \vdots \end{pmatrix}\right] \\ &= \begin{pmatrix} \vdots \\ \sum_{i=0}^{M-1} E\big[w_i^k x_{k-i} x_{k-j}\big] \\ \vdots \end{pmatrix} \\ &= \begin{pmatrix} \vdots \\ \sum_{i=0}^{M-1} \big(w_i^k\big) E[x_{k-i} x_{k-j}] \\ \vdots \end{pmatrix} \\ &= \begin{pmatrix} \vdots \\ \sum_{i=0}^{M-1} \overline{w_i^k} r_{\mathrm{xx}}(i-j) \\ \vdots \end{pmatrix} \\ &= R\overline{W^k} \end{aligned}$$

where $R = E\Big[X^k {X^k}^T\Big]$ is the data correlation matrix.

Putting this back into our equation
Equation:

$$\begin{aligned} \overline{W^{k+1}} &= \overline{W^k} + 2\mu P + -\left(2\mu R\overline{W^k}\right) \\ &= I\overline{W^k} + 2\mu P \end{aligned}$$

Now **if** $\overline{W^{k \to \infty}}$ converges to a vector of finite magnitude ("convergence in the mean"), what does it converge to?

If $\overline{W^k}$ converges, then as $k \to \infty$, $\overline{W^{k+1}} \simeq \overline{W^k}$, and

$$\overline{W^\infty} = I\overline{W^\infty} + 2\mu P$$

$$2\mu R\overline{W^\infty} = 2\mu P$$

$$R\overline{W^\infty} = P$$

or

$$\overline{W_{\text{opt}}} = R^{-1}P$$

the Wiener solution!

So the LMS algorithm, **if** it converges, gives filter coefficients which on average are the Wiener coefficients! This is, of course, a desirable result.

First-order stability

But does $\overline{W^k}$ converge, or under what conditions?

Let's rewrite the analysis in term of $\overline{V^k}$, the "mean coefficient error vector" $\overline{V^k} = \overline{W^k} - W_{\text{opt}}$, where W_{opt} is the Wiener filter

$$\overline{W^{k+1}} = \overline{W^k} - 2\mu R\overline{W^k} + 2\mu P$$

$$\overline{W^{k+1}} - W_{\text{opt}} = \overline{W^k} - W_{\text{opt}} + - \left(2\mu R\overline{W^k}\right) + 2\mu RW_{\text{opt}} - 2\mu RW_{\text{opt}} + 2\mu P$$

$$\overline{V^{k+1}} = \overline{V^k} - 2\mu R\overline{V^k} + -(2\mu R W_{\text{opt}}) + 2\mu P$$

Now $W_{\text{opt}} = R^{-1}$, so

$$\overline{V^{k+1}} = \overline{V^k} - 2\mu R\overline{V^k} + -\left(2\mu R R^{-1} P\right) + 2\mu P = (I - 2\mu R)\overline{V^k}$$

We wish to know under what conditions $\overline{V^{k\to\infty}} \to \bar{0}$?

Linear Algebra Fact

Since R is positive definite, real, and symmetric, all the eigenvalues are real and positive. Also, we can write R as $Q^{-1}\Lambda Q$, where Λ is a diagonal matrix with diagonal entries λ_i equal to the eigenvalues of R, and Q is a unitary matrix with rows equal to the eigenvectors corresponding to the eigenvalues of R.

Using this fact,

$$V^{k+1} = \left(I - 2\mu\left(Q^{-1}\Lambda Q\right)\right)V^k$$

multiplying both sides through on the left by Q: we get

$$Q\overline{V^{k+1}} = (Q - 2\mu\Lambda Q)\overline{V^k} = (1 - 2\mu\Lambda)Q\overline{V^k}$$

Let $V^{'} = QV$:

$$V^{'k+1} = (1 - 2\mu\Lambda)V^{'k}$$

Note that $V^{'}$ is simply V in a rotated coordinate set in $\mathbb{R}^m$, so convergence of $V^{'}$ implies convergence of V.

Since $1 - 2\mu\Lambda$ is diagonal, all elements of $V^{'}$ evolve independently of each other. Convergence (stability) bolis down to whether all M of these scalar, first-order difference equations are stable, and thus $\to (0)$.

$$\forall i, i = [1, 2, \ldots, M] : \left(V_i^{'k+1} = (1 - 2\mu\lambda_i)V_i^{'k}\right)$$

These equations converge to zero if $|1 - 2\mu\lambda_i| < 1$, or $\forall i : (|\mu\lambda_i| < 1)$ μ and λ_i are positive, so we require $\forall i : \left(\mu < \frac{1}{\lambda_i}\right)$ so for convergence in the mean of the LMS adaptive filter, we require
Equation:

$$\mu < \frac{1}{\lambda_{\max}}$$

This is an elegant theoretical result, but in practice, we may not know $\lambda_{\max}$, it may be time-varying, and we certainly won't want to compute it. However, another useful mathematical fact comes to the rescue...

$$\operatorname{tr}(R) = \sum_{i=1}^{M} r_{\mathrm{ii}} = \sum_{i=1}^{M} \lambda_i \geq \lambda_{\max}$$

Since the eigenvalues are all positive and real.

For a correlation matrix, $\forall i, i \in \{1, M\} : (r_{\mathrm{ii}} = r(0))$. So $\operatorname{tr}(R) = Mr(0) = ME[x_k x_k]$. We can easily estimate $r(0)$ with $O(1)$ computations/sample, so in practice we might require

$$\mu < \frac{1}{\widehat{Mr(0)}}$$

as a conservative bound, and perhaps adapt μ accordingly with time.

Rate of convergence

Each of the modes decays as

$$(1 - 2\mu\lambda_i)^k$$

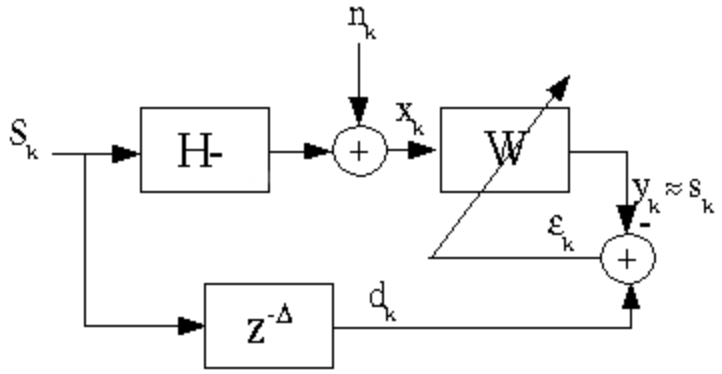
Note: The **initial** rate of convergence is dominated by the fastest mode $1 - 2\mu\lambda_{\max}$. This is not surprising, since a dradient descent method goes "downhill" in the steepest direction

Note: The **final** rate of convergence is dominated by the slowest mode $1 - 2\mu\lambda_{\min}$. For small $\lambda_{\min}$, it can take a long time for LMS to converge.

Note that the convergence behavior depends on the data (via R). LMS converges relatively quickly for roughly equal eigenvalues. Unequal eigenvalues slow LMS down a lot.

Adaptive Equalization

Note: Design an approximate inverse filter to cancel out as much distortion as possible.



In principle, $WH \simeq z^{-\Delta}$, or $W \simeq \frac{z^{-\Delta}}{H}$, so that the overall response of the top path is approximately $\delta(n - \Delta)$. However, limitations on the form of W (FIR) and the presence of noise cause the equalization to be imperfect.

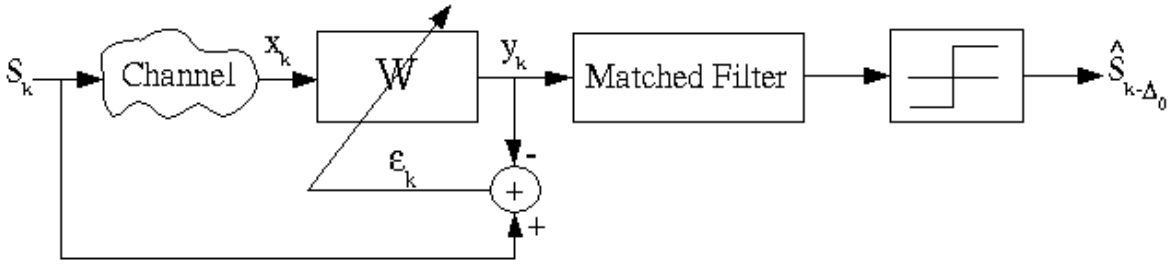
Important Application

Channel equalization in a digital communication system.

If the channel distorts the pulse shape, the matched filter will no longer be matched, intersymbol interference may increase, and the system

performance will degrade.

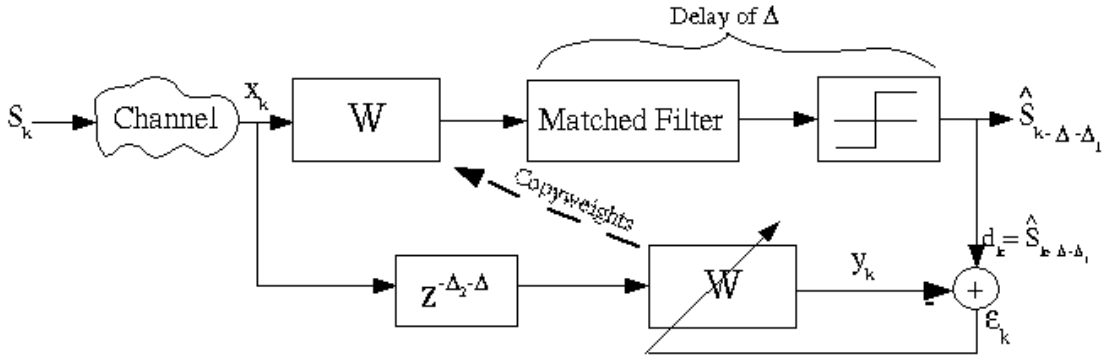
An adaptive filter is often inserted in front of the matched filter to compensate for the channel.



This is, of course, unrealizable, since we do not have access to the original transmitted signal, s_k.

There are two common solutions to this problem:

1. Periodically broadcast a known **training signal**. The adaptation is switched on only when the training signal is being broadcast and thus s_k is known.
2. Decision-directed feedback: If the overall system is working well, then the output $\hat{s}_{k-\Delta_0}$ should almost always equal $s_{k-\Delta_0}$. We can thus use our received digital communication signal as the desired signal, since it has been cleaned of noise (we hope) by the nonlinear threshold device!
   Decision-directed equalizer



As long as the error rate in $\hat{s}_k$ is not too high (say 75%), this method works. Otherwise, d_k is so inaccurate that the adaptive filter can never find the Wiener solution. This method is widely used in the telephone system and other digital communication networks.